AFRL-IF-RS-TR-2004-34
Final Technical Report
February 2004

AN EXCEPTION HANDLING SERVICE FOR SOFTWARE AGENT ENSEMBLES

Massachusetts Institute of Technology

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. G340

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-34 has been reviewed and is approved for publication.

APPROVED: /s/

FRANK H. BORN
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2004	Final Apr 98 – Apr 03

4. TITLE AND SUBTITLE
AN EXCEPTION HANDLING SERVICE FOR SOFTWARE AGENT ENSEMBLES

5. FUNDING NUMBERS
C - F30602-98-2-0099
PE - 63760E
PR - AGEN
TA - T0
WU - 16

6. AUTHOR(S)
Mark Klein

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Massachusetts Institute of Technology
77 Massachusetts Ave
Cambridge Massachusetts 02139

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-34

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Frank H. Born/IFTB/(315) 330-4726/ Frank.Born@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The objective of this project has been to develop technology that enables robust operation in open multi-agent systems (MAS) where we have, in principle, no guarantees of reliable, compliant, or even friendly agents and infrastructures, since we have no control or even necessarily any knowledge over how they are implemented. As part of this project we have developed a substantial web-accessible knowledge base of widely usable exception handling expertise, as well as technology that enables (1) the creation of better contingent contracts between agents (both human and software-based) as well as (2) exception-handling agents that monitor MAS for problem symptoms, diagnose the underlying problems, and intervene as appropriate to avoid or resolve these problems.

14. SUBJECT TERMS
Software Agents, Exception Handling, Multi-Agent Systems

15. NUMBER OF PAGES
123

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

OBJECTIVES 1
TECHNICAL APPROACH 1
OVERVIEW OF RESULTS 1
TECHNICAL ACCOMPLISHMENTS 2
Exception Analysis Methodology *2*
Exception Handling Knowledge Base *3*
Knowledge Base Editing Tools *3*
MAS Design Methodology *3*
Novel Exception Handlers *4*
Contingent Contract Negotiation Algorithms *5*
Exception Handling Agents *5*
KNOWLEDGE DISSEMINATION ACTIVITIES 5
Complete List of Project Publications *5*
Collaborations *9*
Student Theses *10*
Presentations *10*
Spin-offs *10*
REFERENCES 11
APPENDIX: SELECTED PAPERS 12
Appendix A: Using Role Commitment Violation Analysis to Identify Exceptions *13*
Appendix B: Towards a Systematic Repository of Knowledge About Managing Multi-agent System Exceptions. *26*
Appendix C: A Knowledge-Based Methodology for Designing Reliable Multi-Agent Systems *45*
Appendix D: Handling Resource Use Oscillation in Open Multi-Agent Systems *57*
Appendix E: Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death *69*
Appendix F: Protocols for Negotiating Complex Contracts *89*
Appendix G: Exception Handling in Agent Systems *107*

Objectives

The objective of this project has been to develop technology that enables robust operation in open multi-agent systems (MAS) where we have, in principle, no guarantees of reliable, compliant, or even friendly agents and infrastructures, since we have no control or even necessarily any knowledge over how they are implemented. In a coalition operation, for example, we may suddenly find ourselves needing to quickly be able to work with agents from North Korea or Iraq without the luxury of first engaging in an exhaustive code review. The same kinds of issues appear, in less extreme form, in such non-military contexts as electronic commerce, disaster relief, and so on.

Technical Approach

Our technical approach has followed two tracks, both inspired by the ways exceptions are dealt with in human society:

- *Contingent contracts* (improving MAS design): This track has involved developing methodologies and negotiation algorithms that allow humans and software agents to define more ‘exception-proof” contracts that specify what each agent does under normal circumstances (their coordination mechanisms) as well as what they do when things go wrong (their exception handlers).
- *Exception handling agents* (improving MAS operation): We have defined, implemented and evaluated exception handling (EH) agents that monitor a MAS for possible exception symptoms, diagnose the underlying problem when symptoms occur, and intervene to prevent or resolve the exceptions once diagnosed. This idea has been inspired by the many institutions that serve that role in human society, including the police, law courts, security and exchange commission, and so on.

Both of these tracks build upon a knowledge base of generic exception handling expertise, which allows humans and software agents to understand the possible kinds, causes, and responses for a wide range of possible MAS exceptions.

Overview of Results

It is difficult to fully capture the results of over four years of intensive and varied effort by many participants, but in the following we will provide an overview of the key technical and knowledge-dissemination outputs of this project.

Our technical accomplishments have included:

- Developing an *exception analysis methodology* for systematically identifying and organizing MAS exceptions and responses in a way that enables one to find and apply this expertise quickly in new contexts.
- A substantive *knowledge base* of EH knowledge, created by using this methodology to organize, as well as augment, previous work in the MAS and related research literatures
- A collaborative web-based *knowledge base editing tool* for maintaining this knowledge base, that is now in daily use.
- A *MAS design methodology* that exploits the EH knowledge base to help people design more robust MAS coordination mechanisms
- The design, implementation and evaluation of *novel exception handlers* for handling agent death and resource use oscillation
- The design, implementation and evaluation of novel *negotiation algorithms* that enable software agents and humans to more effectively define complex contingent contracts
- The design, implementation and evaluation of *EH agents* that exploit the EH knowledge base to allow run-time monitoring, exception detection, diagnosis and intervention while making minimal requirements of the other MAS agents

Our knowledge-dissemination activities have included over 40 publications, invited presentations, student theses, collaborations with other researchers, and spin-off projects.

Technical Accomplishments

Exception Analysis Methodology

The first step involved in developing more robust multi-agent systems is a systematic approach for identifying and organizing information about what kinds of things can go wrong, and what can be done about them, in a way that facilitates reuse in a wide variety of contexts. Our approach to this challenge is documented in:

Klein, M., *Using Role Commitment Violation Analysis to Identify Exceptions in Multi-Agent Coordination Mechanisms*. Working Paper ASES-WP-2000-04. Center for Coordination Science, Massachusetts Institute of Technology, Cambridge MA USA. 2000. (Available in Appendix)

Klein, M. and C. Dellarocas, *Towards a Systematic Repository of Knowledge about Managing Multi-Agent System Exceptions*. Working paper ASES-WP-2000-01. Center for Coordination Science, Massachusetts Institute of Technology, Cambridge MA USA. 2000. (Available in Appendix).

Exception Handling Knowledge Base

The exception handling knowledge base continues to grow and currently represents over 600 MAS exception types as well as over 300 different exception handler techniques. This expertise has been harvested from previous relevant research literature as well as from the application of our exception analysis methodology to key MAS coordination mechanisms. There are three main classes of coordination possible in MAS, including sharing (resource allocation), fit (collaborative synthesis), and flow (input/output sequences) (Malone and Crowston 1994). We have captured EH expertise for the MAS mechanisms most widely used in all three of these categories, with a particular focus on markets (i.e. auctions), collaborative design, and supply chains. The exceptions we have identified fall into two main categories, including 'commitment violations' (where some agent does not perform a task it had committed to do) as well as 'emergent dysfunctions' (where many locally correct actions combine to produce globally dysfunctional dynamics such as large resource use oscillations).

This knowledge base can be viewed using the WWW-based knowledge-base browser described in the next section.

Knowledge Base Editing Tools

We have gone through several generations of tools for browsing and editing this EH knowledge base, all created building on the ideas developed for the MIT Process Handbook (Malone, Crowston et al. 1999). This work has culminated in the creation of a web-based system that allows multiple users to simultaneously browse and even edit the contents of the EH knowledge base.

This system can be accessed by going to http://franc2.mit.edu/pql/ and logging in with login "guest" and password "guest". Note that this login will only allow users to utilize the browsing, but not the editing, capabilities of the system.

MAS Design Methodology

We have developed a methodology that uses the EH knowledge base to allow human MAS designers to anticipate and design in responses for the exceptions that can occur in such systems. Since there is a very wide range of potential failure modes and ways of

dealing with them, and it is difficult for any one individual or group to know all the expertise applicable to any given system, such a methodology can be very valuable. For further details, see:

M. Klein. *A Knowledge-Based Methodology for Designing Reliable Agent Systems*. Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, held in conjunction with the International Conference on Software Engineering. 2003. Portland, Oregon - USA. (Available in Appendix).

Novel Exception Handlers

In addition to systematizing exception expertise described in the existing research literature, we have also developed novel techniques for several exception types which we have judged to be important and inadequately addressed to date, including:

Collusive reputation fraud. Reputation servers are the exception handling mechanism most widely used in market mechanisms, but are vulnerable themselves to the exception of collusive reputation fraud (wherein agents collude to fraudulently improve or weaken their own or other agent's reputations). Our handlers use selective anonymity and reputation cluster based filtering to substantially mitigate these problems. See: Dellarocas, C. *Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior*. In the *Proceedings of the 2nd ACM Conference on Electronic Commerce*. 2000. Minneapolis, MN.

Resource use oscillation. When resources are allocated on a first-come first-serve basis (as is true in a very wide range of contexts ranging from Internet bandwidth to grocery store checkout lines), delays in the resource status information used by the resource consumers can result in the emergence of large and harmful resource use oscillations. We have developed a novel approach to this problem based on the use of selective stochastic load-dependent resource request rejection. See: Klein, M. and Y. Bar-Yam. *Handling Resource Use Oscillation in Multi-Agent Markets*. in *AAMAS Workshop on Agent-Mediated Electronic Commerce V*. 2003. Melbourne Australia. (Available in Appendix).

Agent death. While good techniques (e.g. rollbacks and mirroring) have been developed for failure recovery in closed distributed systems, comparably effective techniques have not been available for open agent systems where cooperation of the individual components is not guaranteed. Our approach to this challenge integrates a range of techniques, including reputation-based bid filtering, early agent death detection, result caching and task retry/task cancellation interventions. See: Klein, M., J.A. Rodriguez-Aguilar, and C. Dellarocas, *Using Domain-Independent Exception Handling Services to*

Enable Robust Open Multi-Agent Systems: The Case of Agent Death. Autonomous Agents and Multi-Agent Systems, 2003. **7**(1/2). (Available in Appendix).

Contingent Contract Negotiation Algorithms

In an open MAS, we can not assume that the normative and exception-handling behaviors of all of the participating agents will be fixed and universally accepted. Rather, agents must be able to *negotiate* contingent contracts between themselves that specify who will do what. Work to date on negotiation has focused almost exclusively on 'simple' contracts consisting of a single issue (usually price) or several independent issues. Contingent contracts, by contrast, are much more complex, consisting of many inter-dependent issues. This issue inter-dependency radically changes the nature of the algorithms that are suitable for negotiating good contracts. We have developed and evaluated what we believe are the first MAS negotiation algorithms suitable for complex contracts with multiple interdependent issues. For details, see: Klein, M., et al., *Protocols for Negotiating Complex Contracts.* IEEE Intelligent Systems, 2003. In press. (Available in Appendix).

Exception Handling Agents

We have developed an architecture wherein a set of specialized EH agents monitor a diverse community of agents in an open MAS in order to detect exception symptoms, diagnose the underlying cause(s), and intervene when necessary. These EH agents make only minimal assumptions about the other agents in the MAS, requiring only that each agent implement a simple API. This architecture has been implemented and tested extensively using a MAS testbed we developed called SimHazard, and has also been implemented as a Java-based CoABS Grid service. For details, see:

Klein et al. Exception Handling in Agent Systems. Proceedings of the Third International Conference on Autonomous Agents, Seattle, Washington, 1999. (Available in Appendix).

Klein et al. *The Case of Agent Death.* Autonomous Agents and Multi-Agent Systems, 2003. **7**(1/2). (Available in Appendix).

Knowledge Dissemination Activities

Complete List of Project Publications

- Dellarocas, C. and M. Klein. *A Knowledge-Based Approach for Handling Exceptions in Business Processes*. in *Proceedings of the 8th Workshop on Information Technologies and Systems (WITS'98)*. 1998. Helsinki, Finland.

- Klein, M., *A Knowledge-Based Approach to Handling Exceptions in Workflow Systems*. 1998, MIT Center for Coordination Science: Cambridge MA USA.

- Klein, M. and C. Dellarocas. *Exception Handling in Collaborative Web-Based Agents*. in *Proceedings of the Seventh Workshop-Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE)*. 1998. Stanford University, USA.

- Klein, M., *A Knowledge-Based Approach to Handling Exceptions in Workflow Systems*. 1998, MIT Center for Coordination Science: Cambridge MA USA.

- Klein, M. *Toward Adaptive Workflow Systems*. in *Proceedings of a Workshop at the ACM Computer-Supported Collaborative Work Conference*. 1998. Seattle WA.

- Dellarocas, C. and M. Klein. *Towards civil agent societies: Creating robust, open electronic marketplaces of contract net agents*. in *Proceedings of the International Conference on Information Systems (ICIS-99)*. 1999. Charlotte, North Carolina USA.

- Klein, M. and C. Dellarocas. *Exception Handling in Agent Systems*. in *Proceedings of the Third International Conference on AUTONOMOUS AGENTS (Agents '99)*. 1999. Seattle, Washington.

- Dellarocas, C. and M. Klein. *An Experimental Evaluation of Domain-Independent Fault Handling Services in Open Multi-Agent Systems*. in *Proceedings of The International Conference on Multi-Agent Systems (ICMAS-2000)*. 2000. Boston, MA.

- Dellarocas, C., M. Klein, and J.A. Rodriguez-Aguilar. *An Exception-Handling Architecture for Software Agent Marketplaces based on the Contract Net Protocol*. in *ACM Conference on Electronic Commerce 2000*. 2000. Minneapolis, Minnesota, USA.

- Dellarocas, C. and M. Klein, *A knowledge-based approach for handling exceptions in business processes.* Information Technology & Management, 2000. **1**(3): p. 155-69.

- Klein, M. and C. Dellarocas, *A Knowledge-Based Approach to Handling Exceptions in Workflow Systems.* Journal of Computer-Supported Collaborative Work. Special Issue on Adaptive Workflow Systems., 2000. **9**(3/4).

- Klein, M. *Towards a Systematic Repository of Knowledge About Managing Collaborative Design Conflicts*. in *Proceedings of the International Conference on AI in Design (AID-2000)*. 2000. Boston MA: Kluwer Academic Publishers.

- Klein, M. and C. Dellarocas, *Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems*. 2000, Massachusetts Institute of Technology: Cambridge MA USA.
- Klein, M. and C. Dellarocas, *Towards a Systematic Repository of Knowledge about Managing Multi-Agent System Exceptions*. 2000, Massachusetts Institute of Technology: Cambridge MA USA.
- Klein, M. and C. Dellarocas, *Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems*. 2000, Center for Coordination Science, Massachusetts Institute of Technology, Cambridge MA USA.
- Klein, M. and C. Dellarocas, *A Systematic Repository of Knowledge About Handling Exceptions in Business Processes*. 2000, Center for Coordination Science, Massachusetts Institute of Technology, Cambridge MA USA.
- Klein, M., *Using Role Commitment Violation Analysis to Identify Exceptions in Multi-Agent Coordination Mechanisms*. 2000, Center for Coordination Science, Massachusetts Institute of Technology, Cambridge MA USA.
- Klein, M., J.A. Rodriguez-Aguilar, and C. Dellarocas, *Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death*. 2000, Massachusetts Institute of Technology: Cambridge MA USA.
- Woolf, B.P., et al. *A Digital Market Place for Education*. in *Proceedings of the SSGRR 2000 Computer and E-Business Conference*. 2000. L'Aguila Rome Italy.
- Woolf, B.P., et al., *A Digital Market Place for Education*, in *The Internet and Education*, V. Milutinovic and F. Patricelli, Editors. 2000: L'Aguila Rome Italy.
- Faratin, P. and M. Klein. *Automated Contract Negotiation and Execution as a System of Constraints*. in *The IJCAI-01 Workshop on Distributed Constraint Reasoning*. 2001. Seattle WA USA.
- Klein, M., et al., *Negotiating Complex Contracts*. 2001, Massachusetts Institute of Technology: Cambridge MA USA.
- Klein, M., et al. *Negotiating Complex Contracts*. in *AAAI Fall Symposium on Autonomous Negotiating Systems*. 2001. Falmouth, MA, USA: AAAI Press.

- Klein, M., et al. *What Complex Systems Research Can Teach Us About Collaborative Design*. in *International Workshop on CSCW in Design*. 2001. London. Ontario, Canada: IEEE Press.

- Klein, M. and Y. Bar-Yam, *Handing Emergent Dysfunctions in Open Peer-to-Peer Systems*. 2001, MIT Center for Coordination Science: Cambridge MA.

- Faratin, P., et al., *Simple Negotiating Agents in Complex Games: Emergent Equilibria and Dominance of Strategies*, in *Intelligent Agent VIII: Agent Theories, Architectures, and Languages*. 2002, Springer Verlag. p. 367--377.

- Klein, M., et al., *A Complex Systems Perspective on Computer-Supported Collaborative Design Technology*. Communications of the ACM, 2002. **45**(11): p. 27-31.

- Klein, M., et al. *Negotiating Complex Contracts*. in *Autonomous Agents and Multi-Agent Systems*. 2002. Bologna Italy: AAAI Press.

- Klein, M., C. Dellarocas, and J.A. Rodriguez-Aguilar. *A Knowledge-Based Methodology for Designing Robust Multi-Agent Systems*. in *Autonomous Agents and Multi-Agent Systems*. 2002. Bologna Italy: AAAI Press.

- Klein, M., P. Faratin, and Y. Bar-Yam. *Using an Annealing Mediator to Solve the Prisoners' Dilemma in the Negotiation of Complex Contracts*. in *Agent-Mediated Electronic Commerce (AMEC-IV) Workshop*. 2002. Bologna Italy: Springer.

- Klein, M., et al. *A Complex Systems Perspective on How Agents Can Support Collaborative Design*. in *Workshop on Agents in Design*. 2002. Cambridge MA: Key Centre iof Design Computing and Cognition, University of Sidney, Australia.

- Klein, M., P. Faratin, and Y. Bar-Yam, *Using an Annealing Mediator to Solve the Prisoners' Dilemma in the Negotiation of Complex Contracts*, in *Proeedings of the Agent-Mediated Electronic Commerce (AMEC-IV) Workshop*. 2002, Springer-Verlag.

- Klein, M., J.A. Rodriguez-Aguilar, and C. Dellarocas, *Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death.* Autonomous Agents and Multi-Agent Systems, 2003. **7**(1/2).

- Klein, M., et al., *The Dynamics of Collaborative Design: Insights From Complex Systems and Negotiation Research.* Concurrent Engineering Research & Applications, 2003. In press.

- Klein, M., et al., *Negotiating Complex Contracts.* Group Decision and Negotiation, May 2003.

- Klein, M., et al., *A Complex Systems Perspective on Collaborative Design*, in *Multi-Agent Systems: An Application Science*, T. Wagner, Editor. 2003, Kluwer.

- Klein, M. *A Knowledge-Based Methodology for Designing Reliable Multi-Agent Systems*. in *International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, held in conjunction with the International Conference on Software Engineering*. 2003. Portland, Oregon - USA.

- Klein, M. and Y. Bar-Yam. *Handling Resource Use Oscillation in Multi-Agent Markets*. in *AAMAS Workshop on Agent-Mediated Electronic Commerce V*. 2003. Melbourne Australia.

- Klein, M., et al. *Negotiation Algorithms for Collaborative Design Settings*. in *The 10th ISPE International Conference on Concurrent Engineering Research and Applications (CERA-03)*. 2003. Madeira Island, Portugal.

- Parsons, S., J.A. Rodriguez-Aguilar, and M. Klein, *A Bluffer's Guide to Auctions*. 2003, MIT Sloan School of Management: Cambridge MA USA.

- Parsons, S. and M. Klein, *Diagnosing Faults in Open Distributed Systems*. 2003, MIT: Cambridge MA USA.

- Parsons, S. and M. Klein, *Notes on Diagnosis for Open and Distributed Systems*. 2003, MIT: Cambridge MA USA.

- Klein, M., et al., *A Complex Systems Perspective on How Agents Can Support Collaborative Design*, in *Agent Supported Cooperative Work*, Y. Ye and E.F. Churchill, Editors. In press, Kluwer Academic Publishers.

Collaborations

This project has led to fruitful collaborations around the topic of exception handling in multi-agent systems. In addition to the interactions that occurred within the Control of Agent Based Systems (CoABS) program in the context of the Coalition Agent eXperiment (CoAX), Scalability, Control Robustness and Noncombatent Evacuation Operation (NEO) Technology Integration Experiment (TIE), the project has been carried forth through joint research with:

- Dr. Juan Antonio Rodriguez Aguilar, AI Institute, Spain
- Prof. Chrysanthos Dellarocas, MIT

- Prof. Benjamin Grosof, MIT
- Dr. Peyman Faratin, MIT
- Prof. Simon Parsons, Brooklyn CUNY
- Prof. Hiroki Sayama, Japan Telecommunications College
- Dr. Yaneer Bar-Yam, New England Complex Systems Institute
- Dr. Richard Metzler, New England Complex Systems Institute

Student Theses

Our project has led to the completion of three Masters' theses on MAS exception handling, all for students in the MIT Department of Electrical Engineering and Computer Science, all in May 1999: David Shue, Lijin Aryananda, Muthitacharoen.

Presentations

Our project team made technical presentations on this work at the 1/99, 6/99, 9/00 and 7/01 CoABS PI meetings. In addition to that, we have given many invited talks and tutorials on the results of this work at venues including:

- (tutorial) Autonomous Agents Conference. Bologna Italy. July 2002.
- (invited talk) Workshop on New Directions in Software Technology. December 2001.
- (invited talk) International Workshop on CSCW in Design. July 2001.
- (invited talk) WET ICE Workshop on Evaluating CSCW. June 2001.
- (tutorial) Autonomous Agents Conference. May 2001.
- (invited talk) Fuji Xerox Inc. August 2000.
- (tutorial) World Computer Congress. August 2000.
- (invited talk) International Workshop on Distributed Systems of Knowledge. July 2000.
- (tutorial) International Conference on Enterprise Information Systems. March 1999.
- (invited talk) National University of Singapore. November 1998.
- (tutorial) Pacific Rim International Conference on Artificial Intelligence. November 1998.
- (tutorial) International Conference on Multi-Agent Systems. July 1998.

Spin-offs

This project has led the creation of other projects that continue to carry this work forward, funded by Hewlett-Packard, and by the National Science Foundation. We were approached by Premonition Inc to commercialize some aspects of our exception handling technology, and had gotten to the point of arranging license terms when the company ran out of money and folded as part of the general '.com' bust several years ago.

References

Malone, T. W. and K. Crowston (1994). “The interdisciplinary study of coordination.” ACM Computing Surveys **26**(1): 87-119.

Malone, T. W., K. Crowston, et al. (1999). “Tools for inventing organizations: Toward a handbook of organizational processes.” Management Science **45**(3): 425-443.

Appendix: Selected Papers

Appendix A: **Using Role Commitment Violation Analysis to Identify Exceptions in Open Multi-Agent Systems**

Mark Klein
Center for Coordination Science
Massachusetts Institute of Technology
m_klein@mit.edu

Juan Antonio Rodriguez-Aguilar
Center for Coordination Science
Massachusetts Institute of Technology
jarjar@mit.edu

ABSTRACT

In this paper, we describe a systematic knowledge-based methodology, called role commitment violation analysis, for identifying the failure modes ('exceptions') possible in an open multi-agent system utilizing a given coordination mechanism. Examples of this analysis are presented for auction-based resource-sharing mechanisms.

INTRODUCTION

"open systems ... represent arguably the most important application for multi-agent systems" [1]

Multi-agent systems (MAS), we believe, will increasingly be *open* systems, i.e. systems where the constituent components may vary dynamically and are neither developed nor operated under centralized control [2]. A wide range of important applications such as electronic markets, military and disaster relief operations as well as other kinds of virtual organizations, require the kind of instant operability of independently developed components that multi-agent systems have the potential to provide [3] [1].

A critical challenge to realizing this potential is knowing how to develop effective multi-agent systems out of the diverse and unreliable (buggy, malicious, or simply "dumb") agents and infrastructures we can expect to encounter in open contexts The vast majority of MAS work to date has considered well-behaved agents running on reliable infrastructures [4]. It is clear however that in open systems we can expect, in contrast, to find:

- *Unreliable Infrastructures.* In large distributed systems like the Internet, unpredictable host and communication problems can cause agents to slow down or

die unexpectedly, messages to be delayed, garbled or lost, etc. These problems become worse as the applications increase in size and therefore potential points of failure.

- *Non-compliant agents*. In open systems, agents can not always be trusted to follow the rules properly due to bugs, programmer malice and so on. This can be expected to be especially prevalent and important in contexts such as electronic commerce or military operations where there may be significant incentives for fraud or malice.

- *Emergent dysfunctions*. Emerging multi-agent system applications are likely to involve complex and dynamic interactions that can lead to emergent dysfunctions, such as chaotic behavior, with the coordination mechanisms that have proved most popular to date [5] [6].

All of these departures from "ideal" multi-agent system behavior can be called *exceptions*, and can result in poor performance, system shutdowns, and security vulnerabilities.

This paper describes a novel knowledge-based methodology called 'role commitment violation analysis' (RCV) that we have developed to help MAS designers systematically identify the kinds of exceptions that can occur in an open MAS, thereby helping them ensure their design is capable of handling these potential problems. In the remainder of this paper we will describe this methodology using examples drawn from an analysis of exceptions in the auction-based resource sharing mechanisms.

THE RCV METHODOLOGY

The exceptions that characterize a given MAS coordination mechanism can be identified using a technique we developed called Role Commitment Violation (RCV) analysis. RCV analysis is based on the insight that coordination fundamentally involves the process of agents making (implicit or explicit) commitments to each other to make it possible for them to operate effectively given inter-dependencies between their actions [7] [8] [9]. Coordination failures (i.e. exceptions) can then be viewed as the ways in which elements of a MAS can fail to achieve their commitments to each other. RCV analysis systematically identifies such exceptions as follows:

- Identify the *roles* involved in the MAS.
- For each role, identify the *commitments* that each role requires of the other roles.
- For each commitment, identify the ways the commitment may be *violated*

This analysis can be done, of course, starting from scratch for each new coordination mechanism, but we have found it is possible to capture generic roles, commitments and

exceptions in a knowledge base such that the analysis of a new coordination mechanism can be greatly sped up by examining abstract role-commitment-violation patterns that subsume (parts of) the mechanism. This feature makes the RCV analysis technique knowledge-based and therefore cumulative. We will make these ideas more concrete by looking at examples of RCV analysis for auction-based mechanisms below. We will begin by demonstrating a from-scratch analysis, and then demonstrate how an appropriately structured knowledge base can facilitate this kind of analysis.

Role Identification: Our initial examples will concern the well-known auction-based task-sharing mechanism known as the "Contract Net" (CNET) [10] [11] [12] [13] [14]:

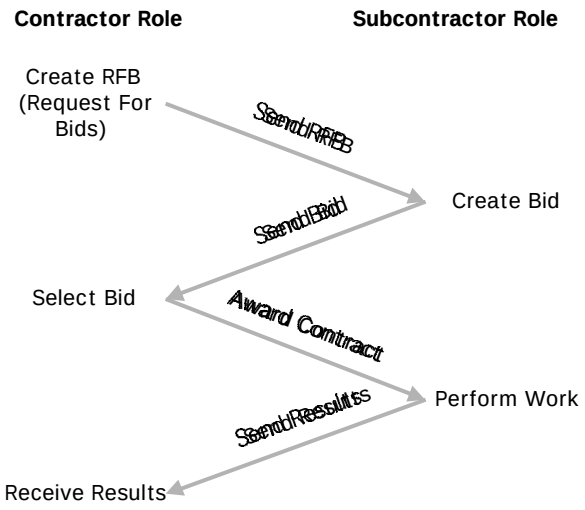


Figure 1. The Contract Net Coordination Mechanism.

An agent (the "contractor") identifies a task that it cannot or chooses not to do locally and attempts to find another agent (the "subcontractor") to perform the task. It begins by creating a Request For Bids (RFB) which describes the desired work, and then sending it to potential subcontractors (typically identified using a matchmaker that indexes agents by the skills they claim to have). Interested subcontractors respond with bids (specifying such issues as the time needed to perform the task) from which the contractor selects a winner. This is thus essentially a first-price sealed-bid auction. The winning agent, once notified of the award, performs the work (potentially subcontracting out its own subtasks as needed) and submits the results to the contractor. We can see that there are thus at least three key roles: the contractor, subcontractor, and matchmaker.

Commitment Identification: The next step is to identify inter-role commitments. As in previous efforts to formalize commitments, every commitment is modeled as involving two parties: the 'source' role requiring the commitment, and the 'target' role being asked to discharge the commitment [7]. There are in general two kinds of commitments: *design*-time commitments that are part of the mechanism definition (e.g. that a bidding agent will send only one bid at a given price level), as well as *run*-time commitments created as a result of the operation of the mechanism (e.g. that an agent will in fact perform the task it was allocated with the contracted cost quality and delivery time).

The first commitment in CNET arises when the contractor queries the matchmaker to get the list of agents potentially appropriate for performing the subtask:

C1: *matchmaker* provides *contractor* with correct candidates in a timely way

In order for the matchmaker to discharge this commitment, it in turn requires commitments on its own, including:

C2: *contractor* provides *matchmaker* with correct skills requirements

C3: *subcontractors* provide *matchmaker* with up-to-date skills descriptions

C4: *messaging-mechanism* provides *matchmaker* with timely and correct delivery of results to contractor

C5: *host-computer* provides *matchmaker* with sufficient computing cycles

C6: *coder* provides *matchmaker* with correct code

Each of these commitments, in turn, may imply further commitments that must be honored in order for them to be discharged correctly. For example, in order to discharge commitment C5, a host computer requires that:

C7: *system-operator* ensures that *host* has sufficient available capacity

We can continue this recursive process of analyzing what commitments are required by the different roles until we produce an exhaustive enumeration thereof.

Just from this simple example we can see several important features of RCV analysis. We are able to perform this analysis based solely on an understanding of the agent-matchmaker interaction protocol, without needing to know any of the details of how this protocol is implemented by any particular agent. The process of enumerating what commitments are required for others to be successfully discharged allows us to identify commitments and even roles that are often left implicit in MAS protocol descriptions (e.g the messaging-mechanism and system-operator roles) leading in principle to a complete description of the commitment set, though in practice it may be quite time-consuming. It

is important, however, that commitments be enumerated *from the idealized perspective of each role.* For example, in an auction the seller ideally wants the following commitment:

C8: *bidders* provide *seller* with bids up to their true valuation for good

even though in many situations this may not be realistic (e.g. in auctions where true value-revelation is not the dominant strategy, or where bidders collude). Many important MAS exceptions represent violations of such idealized commitments.

Commitment Violation (Exception) Identification: The next step of RCV analysis is identifying, for each inter-role commitment, how that commitment can be violated, i.e. what its characteristic exceptions are. There are three major kinds of exceptions:

1. *Infrastructure* exceptions, which represent violations of commitments made by the infrastructure to provide dependable communication and computation. Examples include crashed host computers, and unreliable or slow communication links.
2. *Agent* exceptions, which represent violations of commitments agents make to each other. Examples include agents performing tasks late, doing sub-quality work, or not adhering to the agreed-upon coordination protocols.
3. *Systemic* exceptions, which represent violations of commitments made by the system operator to create a MAS environment well-suited to the tasks at hand. Examples include not populating the MAS with agents containing all the skills needed to perform the tasks at hand, or utilizing coordination mechanisms that produce emergent dysfunctions, i.e. unwelcome dynamic properties such as resource allocation thrashing (when agents spend a disproportionate amount of time re-allocating resources among them) [15] and resource poaching (occurs when a slew of low-priority tasks ties up scarce resources so they are unavailable for later, higher-priority tasks) [16].

Many exceptions can be identified simply as the possible negations of the commitment itself. For example, the *message-sender* role is responsible for discharging three commitments: delivering the *right message* to the *right place* at the *right time*. There are thus three main exceptions: wrong message (the message is garbled), wrong place (the message is delivered to other than the intended target), and wrong time (the message is either too early or, more likely, excessively delayed). As another example, the subcontractor commitment to perform a task with a given quality cost and duration can be violated by late results, sub-contractual quality, or higher-than-produced cost (it is logically possible, though unlikely, that early results, unexpectedly high quality or low costs could also be exceptions).

Unfortunately, it is not possible to exhaustively identify all possible commitment violations in an open MAS via deduction from the commitment descriptions. For

example, commitment C5 above (*host-computer* provides *matchmaker* with sufficient cycles) could be violated because the host computer has experienced a software virus or distributed denial of service attack. Before such attacks were invented, we could not expect to simply infer them from the commitment description. Another example is emergent dysfunctions like those mentioned above. There is as yet no complete analytic theory concerning which kinds of MAS are apt to face which kinds of dynamic dysfunctions. This is a fundamental property of systems, like an open MAS, that are built from black box components, and introduces a experiential component into exception analysis; one must rely on one's previous experience in order to be able to fully identify possible exceptions, and an exhaustive identification may not be possible.

Using a Knowledge Base to Facilitate RCV Analysis: The fact that exception identification can be time-consuming as well as at least partly experiential in nature has led us to explore whether it is possible to 'cache' RCV results so we can analyze exceptions in new protocols more quickly and completely. We have found that we can do so by identifying abstract RCV patterns that appear repeatedly in many different coordination mechanisms. One simple example of this is a generalization of the matchmaker query described above (exceptions are left out of this description for conciseness):

C10: *agent1* provides *agent2* with correct query

C11: *agent2* provides *agent1* with correct response

C12: *messaging-mechanism* provides *agent2* with timely and correct delivery of results

C13: *host-computer* provides *agent2* with sufficient computing cycles

C14: *coder* provides *agent2* with correct code

C15: *system-operator* ensures that *host* has sufficient available capacity

Any query in a coordination mechanism can be viewed as an instance of this pattern, and is subject as a result to the exceptions associated with this pattern.

Our recent efforts have been devoted to identifying a set of abstract RCV patterns suitable for analyzing auctions, building on an abstract auction model (developed by us based on work by [17] [18] [19]) with the following subprocesses:

- Bid call. How buyers are invited to submit bids.
- Ask call. How sellers are invited to submit "asks" (desired bids).
- Bid collection. How bids are collected by auctioneer.
- Ask collection. How asks are collected by auctioneer.
- Winner determination. How the buyer-seller match is found.

- Clearing. How the end of the bidding round is determined.
- Information revelation. How/when bid information is revealed to bidders.
- Closing. When to close the auctioning permanently.

All auction types represent different variants of these subprocesses. A typical Dutch auction [20] for example, uses downward bidding for 'bid call', includes no 'ask call' step, determines the winner as the first bid received as the auctioneer calls prices downwards, clears the bidding round when a bid above the reservation price is received or the reservation price is reached, reveals the winning bid (and possibly the winner's identity) after clearing, and resolves ties by selecting randomly one of the bidders involved. A typical multi-unit Japanese auction [21] involves a calls for bids every time the auctioneer raises the price, includes no 'ask call' step, collects bids as the number of units requested by each bidder, all bidders which do not explicitly request the auctioneer to drop out as part of the winning set, does not reveal any information about bidders' bids during bidding, and clears the bidding round when the total demand from bidders matches the supply (number of units of the lot at auction).

Our analysis to date shows that most auction exceptions are inherited from the abstract auction model, with relatively few exceptions specific to a given protocol. All auctions, for example, are potentially liable to the ‘auctioneer agent crashes’ exception. The bid collection step has such potential exceptions as impersonated bids (where a bidder presents a bid as if it came from someone else), multiple bids from a single buyer at a given price (a protocol violation that may represent a denial of service attack) and so on. Similarly, the winner determination step is potentially prone to exceptions that include unsupported bids (where the winner is unable to pay the winning price), or too few bidders (which may reduce the competitiveness of the auction below a level the auctioneer or seller is willing to accept).

In some cases we are able to identify exceptions that apply to an entire subclass of auction protocols. One example concerns information revelation. If bid information is not revealed (e.g. as with closed bid auctions such as the Vickrey [20]) then this opens the possibility of collusion between the buyer and auctioneer in winner determination; since the bidders will not be in a position to detect such collusion [22]. For another example, all ascending price auctions (e.g. English or Japanese protocols) are prone to non-termination of the winner determination step if enough agents (due either to bugs or malice) simply do not ever drop out.

There are some exceptions, however, which are idiosyncratic to a particular auction mechanism. The Japanese auction, for example, relies on each bidder agent to explicitly indicate that it is reducing its demand or dropping out entirely. If an active bidding agent crashes then the auctioneer will still consider its bids active, prices will be driven above

the appropriate level, possibly infinitely (i.e. non-terminating) if the dead agent's last demand exceeded the total supply, and the dead agent may be inappropriately selected as a winner.

Our knowledge base is also being populated with RCV patterns for the techniques used to handle MAS exceptions, since these are clearly an important part of MAS coordination mechanisms. An interesting multi-tiered example of this occurs concerning tie bids in Dutch auctions. A typical response to this exception is to reset the price to a higher level and restart the descending price clock. A potential exception with this technique is infinite bid collisions, wherein two or more agents collide indefinitely at the same price and as a result the auction does not terminate. A typical response to the infinite bid collision exception, in turn, is to terminate the bidding after a pre-determined number of collisions and then rely on the auctioneer to break the tie, for example by selecting the first bid, last bid, or a random bid. This tie breaking mechanism is potentially subject in turn to the 'buyer/auctioneer collusion' RCV pattern discussed above, wherein in the absence of revealing bid information, it is possible for the auctioneer to favor one bidder over others, e.g. by always rewarding tie bids to the favored bidder. This insight represents another example of how a collection of abstract RCV patterns can be helpful in identifying potential exceptions that might not otherwise be obvious.

A second example of an RCV pattern concerning exceptions with exception handling techniques concerns reputation services [23]. Such services are used to avoid 'non-performing agent' exceptions by publicly identifying agents who have under-performed in the past. Reputation services typically assume the following idealized commitments:

C16: *agents* provide all other *agents* with honest ratings

C17: *agents* perform for other *agents* without discrimination i.e. they are no more likely to non-perform for one agent than another

We can easily imagine agents that violate these assumptions, however. Imagine for example a clique of agents that (perhaps fraudulently) provide each other with a large number of highly positive ratings, while selectively non-performing for targeted customers. The average ratings of such agents are apt to remain high, especially if, as is sometimes done, outlier reputation scores are discarded.

CONTRIBUTIONS OF THIS WORK

The motivation underlying this work is that the space of potentially important exceptions in open multi-agent systems is large and often non-obvious and it is valuable as a result to have a systematic technique for identifying them. The Role-Commitment-Violation

(RCV) methodology presented in this paper represents, we believe, a significant and novel contribution towards this goal.

The power of our approach comes from two sources:

- It mandates a systematic enumeration of all the roles and (implicit as well as explicit, idealized as well as realistic) commitments inherent in a MAS using a given coordination mechanism.
- It builds on a taxonomically-structured knowledge base of abstract RCV patterns that facilitates identifying the exception modes for new protocols.

We consider below the contribution the RCV methodology makes in these two areas.

Role-Commitment Enumeration: Existing agent software engineering methodologies [24] [25] [26] as well as software requirements capture methodologies in general [27] [28] discuss the notions of roles and commitments. Our work differs in that it requires a superset of the inter-agent requirements as typically construed by these approaches for the purpose of completeness in exception identification. Our approach considers, for example, idealized commitments from a given roles' perspective that are not achieved by the final code but remain critical for exception analysis. It also considers a wider variety of roles: agent-oriented software engineering methodologies typically consider only roles that have explicit run-time interactions, while RCV extends this to include other roles such as the system operator, messaging infrastructure, computing infrastructures and so on.

RCV Pattern Knowledge Base: Existing exception identification techniques, except for specialized circumstances such as deadlock properties in distributed protocols [29] [30], have left the identification of possible exception modes up to the software designers themselves [31] [32]. Our work differs in that it includes an approach for collecting abstract exception information that is applicable to a range of MAS coordination mechanisms, thereby offering the potential of greatly reducing the effort involved in exception analysis. RCV can also be viewed, we believe, as a useful complement to traditional failure identification techniques well-suited to open agent systems because it focuses on agent inter-relationships rather than failure modes for the agents themselves.

FUTURE WORK

Our work in this area is progressing along several fronts. We are continuing to accumulate a knowledge base of RCV patterns for auction mechanisms. We have focused so far on Dutch, first-price sealed-bid, and Japanese auctions, and will be analyzing important types including Vickrey and N-to-N (e.g. double dutch) auctions.

Our current commitment representation is semi-formal. Formally represented commitment structures can offer additional power, for example as a basis of exception diagnosis services**,** as follows [33] [34] [35]. If a given role has failed to achieve a commitment, and none of the commitments it in turn requires have been violated, then we can infer that the agent implementing the role has failed, otherwise we can trace back through the causal web of the commitment structure to find the responsible element(s). Note that this approach does not violate the black box assumption we must make in open MAS; we make no assumptions about how an agent works but merely observe its actions.

We have found that RCV patterns can be arranged into a taxonomy based on their function, in the same way that books in a library are arranged by topic, in order to facilitate quickly finding the patterns that apply to the coordination mechanism one is interested in. We have begun developing such a taxonomy based on previous efforts from the process management and MAS research communities [7] [36].

ACKNOWLEDGMENTS

This work was supported by NSF grant IIS-9803251 (Computation and Social Systems Program) and by DARPA grant F30602-98-2-0099 (Control of Agent Based Systems Program). The authors gratefully acknowledge many helpful comments from the members of the MIT Center for Coordination Science.

REFERENCES

[1] Wooldridge, M., N.R. Jennings, and D. Kinny. *A Methodology for Agent-Oriented Analysis and Design*. in *Proceedings of the Third Annual Conference on Autonomous Agents (AA-99)*. 1999. Seattle WA USA: ACM Press.

[2] Hewitt, C. and P.D. Jong, *Open Systems*. 1982, Massachusetts Institute of Technology.

[3] Jennings, N.R., K. Sycara, and M. Wooldrige, *A Roadmap of Agent Research and Development.* Autonomus Agents and Multi-Agent Systems, 1998. 1: p. 275-306.

[4] Hägg, S. *A Sentinel Approach to Fault Handling in Multi-Agent Systems*. in *Proceedings of the Second Australian Workshop on Distributed AI, in conjunction with Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96)*. 1996. Cairns, Australia.

[5] Youssefmir, M. and B. Huberman. *Resource contention in multi-agent systems*. in *First International Conference on Multi-Agent Systems (ICMAS-95)*. 1995. San Francisco, CA, USA: AAAI Press.

[6] Sterman, J.D., *Learning in and about complex systems*. 1994, Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts Institute of Technology. 51.

[7] Singh, M.P., *An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts.* Artificial Intelligence and Law, 1999.

[8] Jennings, N.R., *Coordination Techniques for Distributed Artificial Intelligence*, in *Foundations of Distributed Artificial Intelligence*, G.M.P. O'Hare and N.R. Jennings, Editors. 1996, John Wiley & Sons. p. 187-210.

[9] Gasser, L., *DAI Approaches to Coordination*, in *Distributed Artificial Intelligence: Theory and Praxis*, N.M. Avouris and L. Gasser, Editors. 1992, Kluwer Academic Publishers. p. 31-51.

[10] Smith, R.G. and R. Davis, *Distributed Problem Solving: The Contract Net Approach.* Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence, 1978.

[11] Baker, A. *Complete manufacturing control using a contract net: a simulation study*. in *Proceedings of the International Conference on Computer Integrated Manufacturing*. 1988. Troy New York USA: IEEE Computer Society Press.

[12] Boettcher, K., D. Perschbacher, and C. Wessel, *Coordination of distributed agents in tactical situations.* Ieee, 1987(87CH2450): p. 1421-6.

[13] Bouzid, M. and A.-I. Mouaddib, *Cooperative uncertain temporal reasoning for distributed transportation scheduling.* Proceedings International Conference on Multi Agent Systems, 1998.

[14] Smith, R.G. and R. Davis, *Applications Of The Contract Net Framework: Distributed Sensing.* Distributed Sensor Nets: Proceedings of a Workshop, 1978.

[15] Youssefmir, M. and B.A. Huberman, *Clustered volatility in multiagent dynamics.* Journal of Economic Behavior & Organization, 1997. 32(1): p. 101-18.

[16] Chia, M.H., D.E. Neiman, and V.R. Lesser. *Poaching and distraction in asynchronous agent activities*. in *Proceedings of the Third International Conference on Multi-Agent Systems*. 1998. Paris, France.

[17] Sandholm, T., *eMediator: A Next Generation Electronic Commerce Server*. 1999, Washington University at St. Louis: St Louis, MO, USA.

[18] Wurman, P.R., M.P. Wellman, and W.E. Walsh, *The Michigan Internet AuctionBot: a configurable auction server for human and software agents.* Proceedings of the Second International Conference on Autonomous Agents. ACM., 1998.

[19] Rodriguez-Aguilar, J.A., et al. *Competitive Scenarios for Heterogeneous Trading Agents*. in *Second International Conference on Autonomous Agents (AGENTS'98)*. 1998.

[20] McAfee, R.P. and J. McMillan, *Auctions and Bidding.* J. Economics Lit., 1987. XXV: p. 699--738.

[21] Cassady, R., *Auctions and Auctioneering*. 1967: University of California Press.

[22] Priest, C. *Commodity Trading Using An Agent-based Iterated Double-Auction*. in *Third International Conference on Autonomous Agents (AGENTS'99)*. 1999.

[23] Dellarocas, C. *Mechanisms for coping with unfair ratings and discriminatory behavior in online reputation reporting systems*. in *International Conference on Information Systems (ICIS-00) (under review)*. 2000.

[24] Wooldridge, M., N.R. Jennings, and D. Kinny, *The Gaia methodology for agent-oriented analysis and design.* Autonomous Agents & Multi Agent Systems, 2000. 3(3): p. 285-312.

[25] Bussmann, S., N. Jennings, and M. Wooldridge, *On the Identification of Agents*, in *Agent-Oriented Software Engineering*, P. Ciancarini and M. Wooldridge, Editors. Dec 2000 (in press).

[26] Venkatraman, M. and M.P. Singh, *Verifying Compliance with Commitment Protocols: Enabling Open Web-Based Multiagent Systems.* Autonomous Agents and Multi-Agent Systems, 1999. 3(3).

[27] Pohl, K., *The Three Dimensions of Requirements Engineering: A Framework and its Applications.* Information Systems, 1994. 19(4): p. 234-248.

[28] Finkelstein, A. *Requirements Engineering: a review and research agenda*. 1994.

[29] Avresky, D.R. and D.R. Kaeli, *Fault-tolerant parallel and distributed systems*. 1998, Boston, Mass: Kluwer Academic. xii, 399.

[30] Mullender, S.J., *Distributed systems*. 2nd ed. ACM Press frontier series. 1993, New York Wokingham, England ; Reading, Mass.: ACM Press ; Addison-Wesley Pub. Co. xvi, 601.

[31] Hunt, J., D.R. Pugh, and C.J. Price, *Failure mode effects analysis: a practical application of functional modeling.* Applied Artificial Intelligence, 1995. 9(1): p. 33-44.

[32] Raheja, D. *Software system failure mode and effects analysis (SSFMEA)-a tool for reliability growth*. in *Proceedings of the International Symposium on Reliability*

and Maintainability (ISRM-90). 1990. Tokyo, Japan: Union of Japanese Sci. & Eng; Tokyo, Japan.

[33] Klein, M. and C. Dellarocas. *Exception Handling in Agent Systems*. in *Proceedings of the Third International Conference on AUTONOMOUS AGENTS (Agents '99)*. 1999. Seattle, Washington.

[34] Klein, M. and C. Dellarocas, *Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems*. 2000, Massachusetts Institute of Technology: Cambridge MA USA.

[35] Kleer, J.d., A.K. Macworth, and R. Reiter. *Characterizing Diagnoses*. in *Proceedings of AAAI-90*. 1990.

[36] Malone, T.W., et al., *Tools for inventing organizations: Toward a handbook of organizational processes.* Management Science, 1999. 45(3): p. 425-443.

Appendix B: **Towards a Systematic Repository of Knowledge About Managing Multi-agent System Exceptions**

MARK KLEIN, PHD
Center for Coordination Science
Massachusetts Institute of Technology
m_klein@mit.edu
http://ccs.mit.edu/klein/

CHRYSANTHOS DELLAROCAS, PHD
Sloan School of Management
Massachusetts Institute of Technology
dell@mit.edu
http://mit.ccs.edu/dell/

Abstract. A critical challenge to creating effective agent-based systems is allowing them to operate effectively in environments where failures ('exceptions') can occur. An important barrier to achieving this has been the lack of systematized dissemination of exception handling techniques. This paper describes a semi-formal Web-accessible repository, built as an augmentation of the MIT Process Handbook, that is designed to enable learning about, adding to, and exploiting multi-agent system exception handling expertise.

The Challenge

A critical challenge to creating effective agent-based systems is making them robust in the face of potential failures. Most work to date on multi-agent systems has focused, however, on supporting such basic functionality such as matchmaking (Decker, Sycara et al. 1997) and inter-agent communication (Finin, Labrou et al. 1997), and has typically assumed relatively simple closed environments where the infrastructure is reliable and agents can be trusted to work correctly (Hägg 1996). It is clear however that in many if not most important applications for multi-agent technology, these assumptions will not hold. We can expect, in contrast, to find at least somewhat unreliable infrastructures (such as the Internet), non-compliant agents (due to bugs or even malicious intent), and

emergent dysfunctions (when simple coordination mechanisms produce unexpected dysfunctional behaviors in complex contexts). All of these departures from "ideal" multi-agent system behavior can be called *exceptions*, and the results of inadequate exception handling include the potential for poor performance, system shutdowns, and security vulnerabilities.

A key barrier to the development and utilization of improved exception handling in multi-agent systems has been a lack of systematized dissemination of expertise in this area. Exception handling is fundamentally a multi-disciplinary topic with research efforts scattered across disparate communities including real-time systems (Burns and Wellings 1996), distributed systems (Mullender 1993), robotics (Hindriks, de Boer et al. 1998), computer-supported cooperative work (Mi and Scacchi 1993) (Chiu, Karlapalem et al. 1997) (Klein 1998) (Auramaki and Leppanen 1989) (Finkelstein, Gabbay et al. 1994), manufacturing control (Fletcher and Misbah 1999) (Adamides and Bonvin 1993) (Katz 1993), planning (Traverso, Spalazzi et al. 1996) (Howe 1995) (Birnbaum, Collins et al. 1990) (Broverman and Croft 1987) (Firby 1987), and multi-agent systems (Hägg 1996) (Kaminka and Tambe 1997) (Venkatraman and Singh 1999) (Bansal, Ramohanarao et al. 1997)**.** The result is that good ideas developed within one discipline do not readily propagate to researchers and practitioners in other settings, and opportunities are lost to carry on a more systematic and cumulative exploration of the range of potentially useful exception handling techniques.

The work described in this paper addresses these challenges directly by developing a semi-formal Web-accessible repository of exception handling expertise organized so as to facilitate key uses including:

- Pedagogy: helping students, researchers and practitioners learn about the state of the art in exception handling management
- Development: helping practitioners develop more robust multi-agent systems
- Research: helping researchers identify gaps in exception management technology, identify common abstractions, facilitate discussion, and foster development of new ideas

The remainder of this paper will describe the key ideas and tools making up the exception handling repository, evaluate its efficacy with respect to the goals listed above, and describe potential directions for future work.

Our Approach

Our approach is to capture exception handling knowledge using a substantively extended version of the tools and techniques developed as part of the MIT Process Handbook project. The Handbook is a process knowledge repository which has been under development at the Center for Coordination Science (CCS) for the past six years (Malone and Crowston 1994) (Malone, Crowston et al. 1998). The growing Handbook database currently includes over 5000 process descriptions ranging from specific (e.g. for a university purchasing department) to generic (e.g. for resource allocation and multi-criteria decision making). The CCS has developed a Windows-based tool for editing the Handbook repository contents, as well as a Web-based tool for read-only access. The Handbook is under active use and development by a highly distributed group of more than 40 scientists, teachers, students and sponsors for such diverse purposes as adding new process descriptions, teaching classes, and business process re-design.

In the following sections we will present the core concepts underlying the Handbook, describe how these concepts and associated tools were extended to capture exception handling expertise, and discuss how this enables pedagogy, research, and development.

2.1. Underlying Process Handbook Concepts

The Handbook takes advantage of four simple but powerful concepts to capture and organize process knowledge: *attributes, decomposition, dependencies,* and *specialization.*

Process Attributes: Like most process modeling techniques, the Handbook allows processes to be annotated with attributes that capture such information as a textual description, typical performance values (e.g. how long a process takes to execute), as well as applicability conditions (i.e. constraints on the contexts where the process can be used).

Decomposition: Also like most process modeling techniques, the Handbook uses the notion of *decomposition*: a process is modeled as a collection of activities that can in turn be broken down ("decomposed") into subactivities.

Dependencies: Another key concept we use is that coordination can be viewed as the management of *dependencies* between activities (Malone and Crowston 1994). Every dependency can include an associated *coordination mechanism*, which is simply the process that manages the resource flow and thereby coordinates the activities connected by the dependency. The key advantage of representing processes using dependencies and

coordination mechanisms is that they allow us to abstract away details about how 'core' activities coordinate with each other, and thereby making it easier to explore different ways of doing so. We will see examples of this below.

Specialization: The final key concept is that processes can be arranged into a *taxonomy*, with very generic processes at one extreme and increasingly *specialized* processes at the other. Processes are organized based on their function, so that processes with similar purposes appear close to each other. This facilitates finding and comparing alternative ways for performing functions of interest, thereby fostering easy transfer of ideas. Sibling processes can be grouped into "bundles" with tradeoff tables that capture the relative pros and cons of these alternatives.

2.2. Extending the Handbook to Capture exception Knowledge

While the Handbook as described above is well-suited for describing exception handling processes by themselves, it does not capture crucial information concerning what *types* of exceptions exist, in what *contexts* (i.e. for which multi-agent coordination mechanisms) they can appear, what *impact* they have, or what processes are suitable for *handling* them. *The novel contribution of the work described herein involved extending the Handbook so it can capture this information*. This required two additional elements: the *exception taxonomy*, and the *exception management meta-process*. These are described below.[1]

Exception Taxonomy: The exception taxonomy is a hierarchy of exception types, ranging from general exceptions on the left to more specific ones on the right (Figure 1):

[1] The examples described below will all refer to the well-known coordination mechanism known as the 'Contract Net' protocol (Smith and Davis 1978), which is used to match up tasks with agents in multi-agent systems. In this protocol, an agent (hereafter called the 'contractor') attempts to find another agent (hereafter called the 'subcontractor') to perform some task. It begins by creating a Request For Bids (RFB) which describes the desired work, and then sending it to potential subcontractors (typically identified using a matchmaker that indexes agents by the skills they claim to have). Interested subcontractors respond with bids (specifying such issues as the time needed to perform the task) from which the contractor selects a winner. The winning agent, once notified of the award, performs the work (potentially subcontracting out its own subtasks as needed) and submits the results to the contractor.

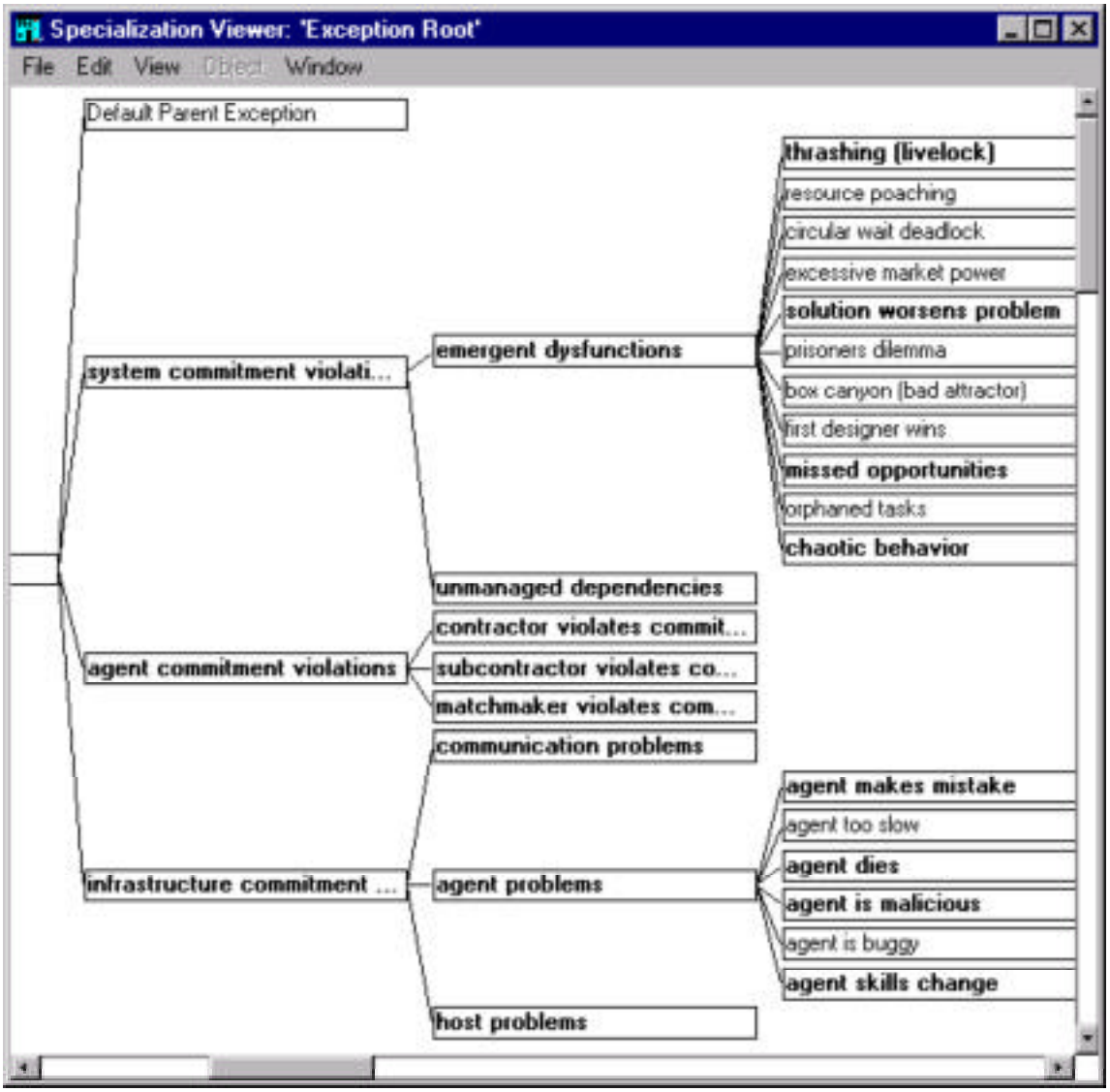


Figure 1. A fragment of the exception type taxonomy.

As the taxonomy shows, there are three main classes of exception possible in multi-agent systems, all of which can be framed as violations of some (implicit or explicit) commitment:

- *Infrastructure Commitment Violations:* The infrastructure has a commitment to provide reliable and predictable agent operation and communication. In large distributed systems like the Internet, however, unpredictable node and link failures may cause agents to die unexpectedly, messages to be delayed, garbled or lost, etc. If a contract net agent dies, for example, there are several immediate consequences. If the agent is acting as a subcontractor, its customer clearly will not receive the results

it is expecting. In addition, if the agent has subcontracted out one or more subtasks, these subtasks and all the sub-sub-… tasks created to achieve them become 'orphaned', in the sense that there is no longer any real purpose for them and they are uselessly tying up potentially scarce subcontractor resources. Finally, if the system uses a matchmaker, it will continue to offer the now dead subcontractor as a candidate (a 'false positive'), resulting in wasted message traffic.

- *Agent Commitment Violations*. The agents in a multi-agent system have a commitment to adhere properly to the coordination protocols they participate in. In open systems, however, agents are developed independently, come and go freely, and thus can not always be trusted to follow the rules properly due to bugs, bounded rationality, programmer malice and so on. This can be expected to be especially prevalent and important in contexts such as electronic commerce where there may be significant incentives for fraud. A non-compliant contract net agent, for example, could wreak havoc through fraudulent advertising, bidding or subcontracting. Imagine, for example, an agent that falsely informs the matchmaker that it has a comprehensive set of skills, sends in highly attractive but fraudulent bids (e.g. specifying it can do any task almost instantaneously) for all pending tasks, and once it wins the awards returns either no results, or simply incorrect ones. The result would be that many if not all of the system's tasks would be awarded to a non-performing agent.

- *System Commitment Violations*. The multi-agent system manager/designer has a commitment to define the system so the agents can get work done efficiently. The relatively lightweight multi-agent coordination mechanisms that have proved popular to date can, however, produce unexpected emergent dysfunctions in complex dynamic contexts. One example is 'resource poaching', wherein a slew of low-priority but long-duration tasks tie up the subcontractors, thereby freezing out resources needed for the higher-priority tasks that arrive later (Chia, Neiman et al. 1998). This does not represent an error per se, but rather an unexpected consequence of the protocol when applied in a complex environment.

Different multi-agent system coordination mechanisms have differing characteristic exception types. The Contract Net protocol, for example, is vulnerable to fraudulent bids, while other protocols like TEAMCORE (Tambe M 1997) have different vulnerabilities (e.g. agents failing to detect conditions that should trigger pre-arranged coordinated actions). The MIT exception repository captures these relationships by building on a taxonomy of multi-agent system coordination mechanisms (Figure 2):

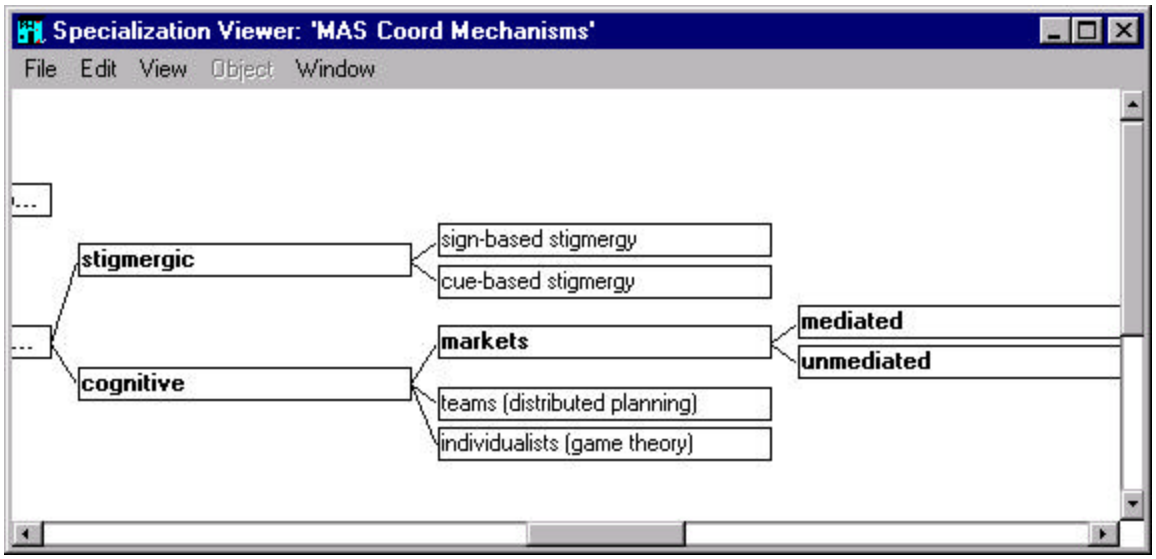


Figure 2. A fragment of the coordination mechanism taxonomy.

Every coordination mechanism is linked to the exception types that characterize it; a mechanisms' characteristic exceptions are inherited by its specializations unless explicitly over-ridden. Every exception is annotated with its typical impact on the associated coordination mechanism. The agent death exception, for example, can as we have seen reduce the efficiency of the multi-agent system by wasting bandwidth and host cycles.

Exception types are linked, in turn, to the exception handling processes suitable for handling them; these processes are themselves arranged into a taxonomy, producing the following overall structure (Figure 3):

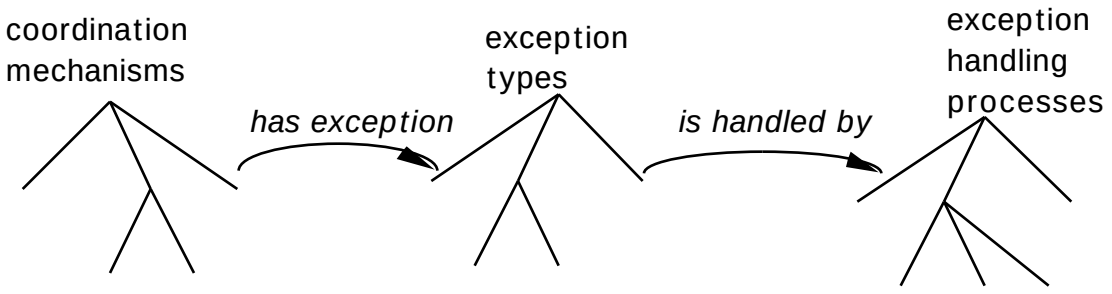


Figure 3. Linkages to/from the exception type taxonomy

The exception handing process taxonomy (see Figure 4) is where the bulk of the repository content resides:

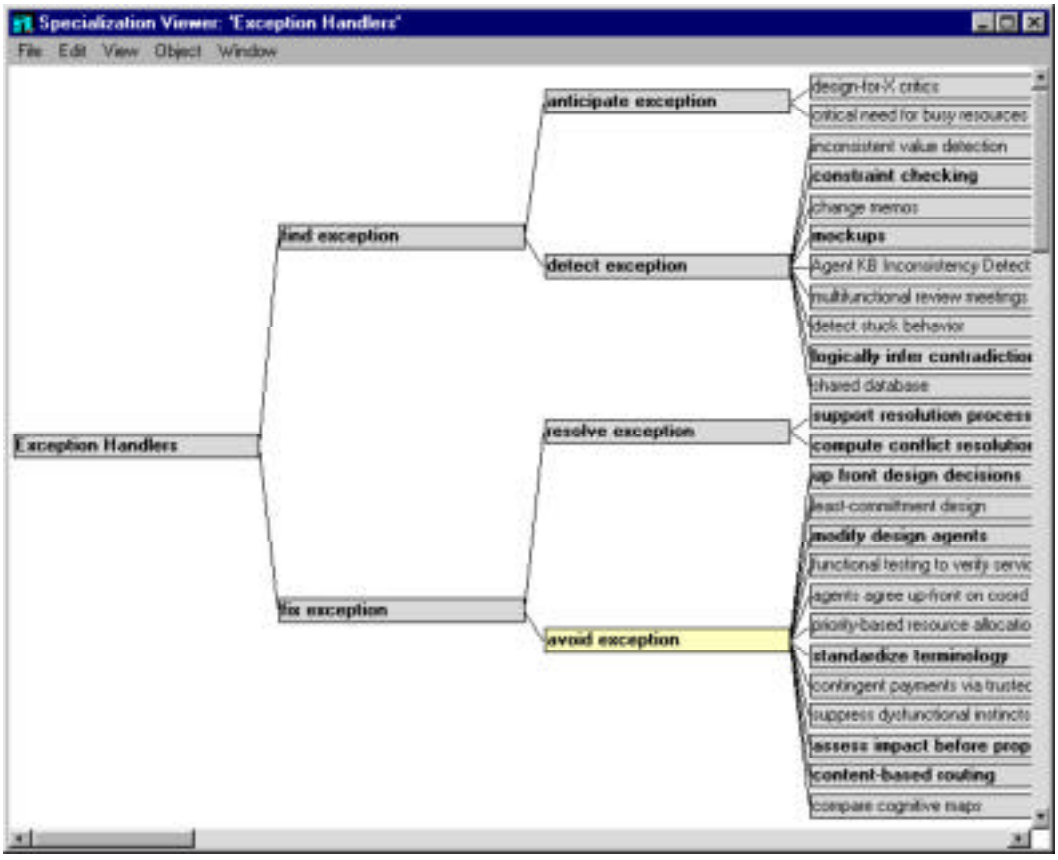


Figure 4. A subset of the exception handling process taxonomy.

There are four main classes of exception handling processes, divided into two pairs. If a exception has not yet occurred, we can use:

- Exception *anticipation* processes, which uncover situations where a given class of exception is likely to occur**.** Resource poaching, for example, can be anticipated when there is a flood of long duration tasks requiring scarce, non-preempting contract net agents to perform them.

- Exception *avoidance* processes, which reduce or eliminate the likelihood of a given class of exception. Resource poaching can be avoided, for example, by allowing contract net agents to preempt currently executing tasks in favor of higher priority pending tasks.

If the exception has already occurred, we can use:

- Exception *detection* processes, which detect when an exception has actually occurred. These can range from simple mechanisms such as timeouts (e.g. to help detect when a subcontractor agent has died) to more sophisticated mechanisms such as socially attentive monitoring (Kaminka and Tambe 1997) wherein agents look for violations of normative relationships with other members of their team.

- Exception *resolution* processes, which resolve an exception once it has happened. A common resolution process is for an agent to simply try again (e.g. re-send an RFB if it received no bids because all the subcontractors were busy last time).

The Exception Management Meta-Process: The exception taxonomy and associated links described above capture the range of *possible* exceptions and associated exception handling processes, but do not specify *which* handlers should be used *when* for *what* exceptions. This latter information is captured in a taxonomy in the exception repository as specializations of the generic *exception management meta-process* (Figure 5):

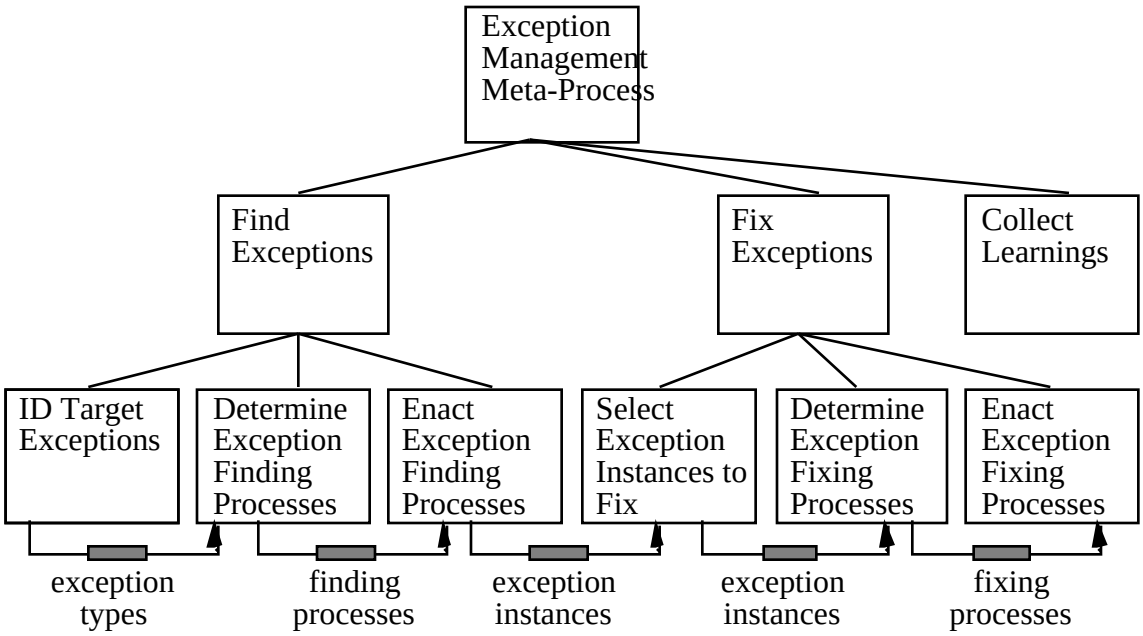


Figure 5. The decomposition of the generic exception management meta-process.

The generic exception management meta-process consists of the following subtasks:

- *Identify target exceptions*, which decides which classes of exceptions the process is going to handle, potentially in a time-varying context-sensitive way.

- *Determine exception finding processes,* which determines which exception finding (i.e. anticipation or detection) handlers will be used to find the exceptions of these types
- *Enact exception finding processes,* which enacts the exception finding processes identified in the previous step, producing one or more exception instances
- *Select exception instances to fix,* which sorts and prunes the list of exception instances so uncovered
- *Determine exception fixing processes,* which determines which exception fixing (avoidance or resolution) processes will be used to handle these exception instances
- *Enact exception fixing processes,* which enacts the exception fixing processes to actually (hopefully) complete the handling of the exception(s) detected by the system
- *Collect learnings,* which collects information produced by any of the other steps as input to any learning capability that the exception management system may have, presumably changing the operation of the other meta-process steps in the future.

All exception management approaches can be captured, we have found, as specializations of this generic process, differing only in how the different subtasks are implemented and connected with each other.

Let us consider two specific examples to make this more concrete. A major distinction among exception handling approaches is whether they rely on up-front mechanism design to prevent exceptions from occurring ('design-time' exception handling), or whether they rely on some kind of run-time detection and remediation to deal with them instead. An example of the first kind of mechanism is the 'levelled commitment protocol', an augmentation of the Contract Net designed to address the 'agents renege on commitment' exception, i.e. where agents unilaterally decide to cancel contracts (Sandholm, Sikka et al. 1999). This protocol relies on a set on de-commitment penalties to entice agents to avoid dysfunctional instances of this exception (Table 1):

Subtask	***How Implemented***
Identify target exceptions	The target exception of this mechanism is selected at design time: 'agent reneges on commitment'.
Determine exception finding processes	The exception finding process is fixed at design-time: "always anticipate this exception".
Enact exception finding processes	The exception finding process is enacted at design time.
Select exception instances to fix	All exceptions are selected.

Determine exception fixing processes	The exception fixing process is fixed at design time to be "modify protocol to use levelled commitments".
Enact exception fixing processes	The exception fixing process is enacted at design time, producing the enhanced Contract Net protocol.
Collect learnings	N/A

TABLE 1. An example of a design-time exception management meta-process.

A run-time exception handling approach being developed by the authors (Dellarocas and Klein 1999) (Klein and Dellarocas 1999) can, by contrast, be described as follows:

Subtask	***How Implemented***
Identify target exceptions	The target exceptions are identified at run time by querying the agents as to which coordination mechanism they will use, and then querying the exception repository to find all the characteristic exceptions for this mechanism.
Determine exception finding processes	Use the exception repository at run time to determine the exception finding processes appropriate for the target exceptions, and then select one or more using the handler process attributes.
Enact exception finding processes	Enact exception finding processes automatically at run time by special agents known as 'sentinels'.
Select exception instances to fix	Select all exception instances as they occur at run time.
Determine exception fixing processes	Use a heuristic classification process at run time to traverse a causal tree to find the root exception (e.g. agent death) underlying the detected exception (e.g. late task), and then picks one of the exception fixing processes linked to the underlying exception, using the handler process attributes.
Enact exception fixing processes	Enact the exception fixing processes automatically at run time by special agents known as 'firefighters'.
Collect learnings	Store exception resolution cases at run time for periodic analysis in order to modify the exception handler process attributes to enable improved exception handler selection.

TABLE 2. An example of a run-time exception management meta-process.

These examples represent two ends of the spectrum of exception management meta-processes. One can of course express a whole range of meta-processes, each with

different characteristic subtasks drawn from their own portion of the repository's process taxonomy. The exception repository process taxonomy includes branches of specializations for every subtask in the generic exception handling meta-process. The 'Identify Target Exception' subtask, for example, includes specializations such as the following:

[1] Select target exceptions at design time to be <list of exceptions> (as above)
[2] Select target exceptions to be all exceptions listed in the repository for the currently active coordination mechanisms (also as above)
[3] Ask the system manager to define the target exceptions at run time
[4] Determine target exceptions automatically in a context-sensitive way, e.g. "only look for fraudulent agent exceptions if the system includes non-certified agents"

This concludes our review of the key concepts underlying the structure of the exception repository. Let us now turn to considering its potential uses.

2.3. Using The Exception Repository

As noted above, we have identified three important potential uses for the exception repository: pedagogy, multi-agent systems development, and research:

Pedagogy: We have developed a Web-accessible read-only version of the Process Handbook that provides access, for pedagogical purposes, to all elements of the exception repository. This tool allows one to view, in outline form, any of the taxonomies (of coordination mechanisms, exception types, exception handler processes and exception management meta-processes) in the repository. One can also bring up a detailed description for any individual process or exception, with buttons and hotlinks that allow one to traverse to all related entities (e.g. from exceptions to the handlers for that exception) (Figure 6):

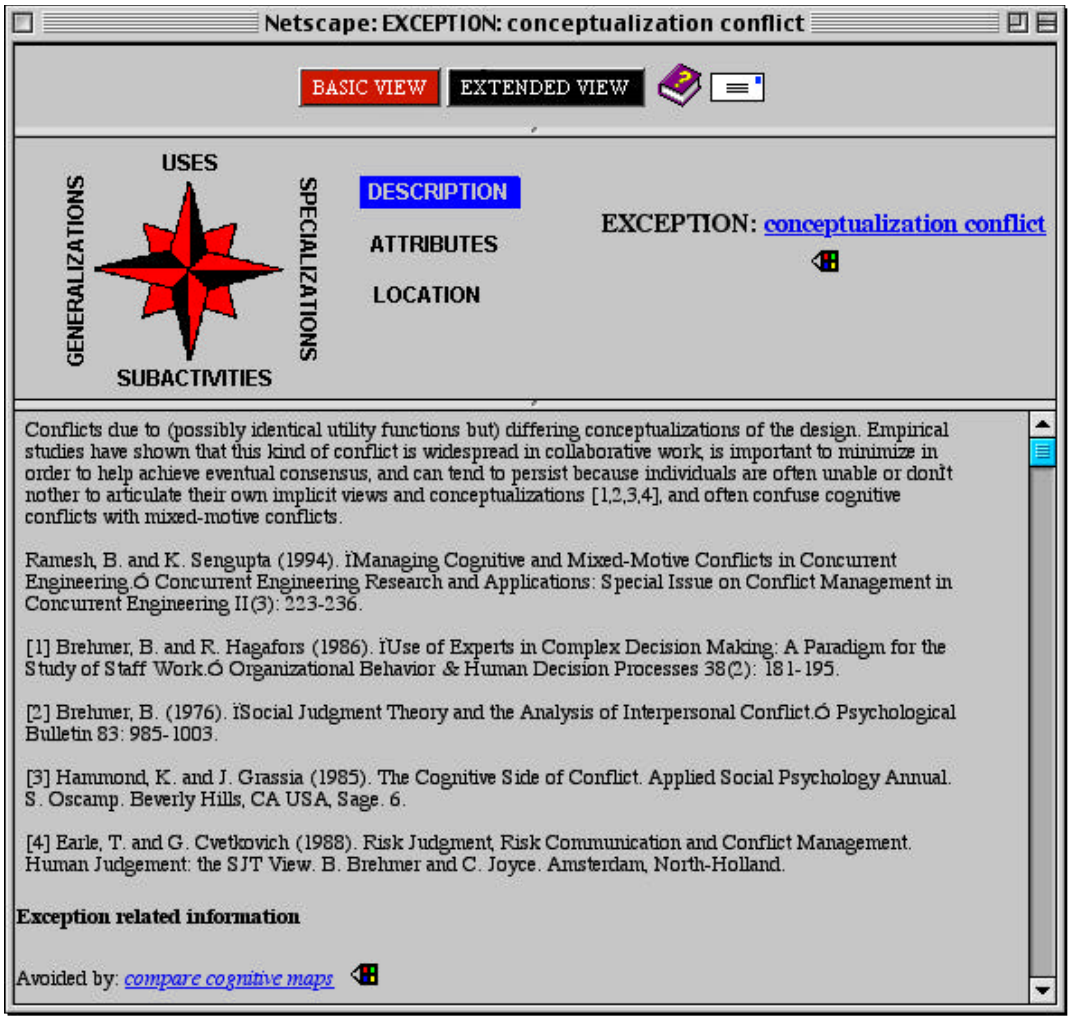


Figure 6: Screen snapshot of the Web-accessible version of the exception repository.

Development: The exception repository supports a simple but powerful methodology for [re-] designing the exception management procedures used in one's multi-agent system. It involves applying the Handbook's process re-design methodology (Herman, Klein et al. 1998) to the exception management meta-processes. All of the subtasks in this process, as we have seen, have multiple alternative specializations (i.e. ways of realizing that subtask). We can therefore explore many different variations of the exception management process by systematically varying the alternatives we select for each subtask**.**

A tool known as the 'Process Recombinator' (Bernstein, Klein et al. 1999), available as part of the Windows version of the Process Handbook, has been developed to support this systematic exploration of different subtask combinations (Figure 7).

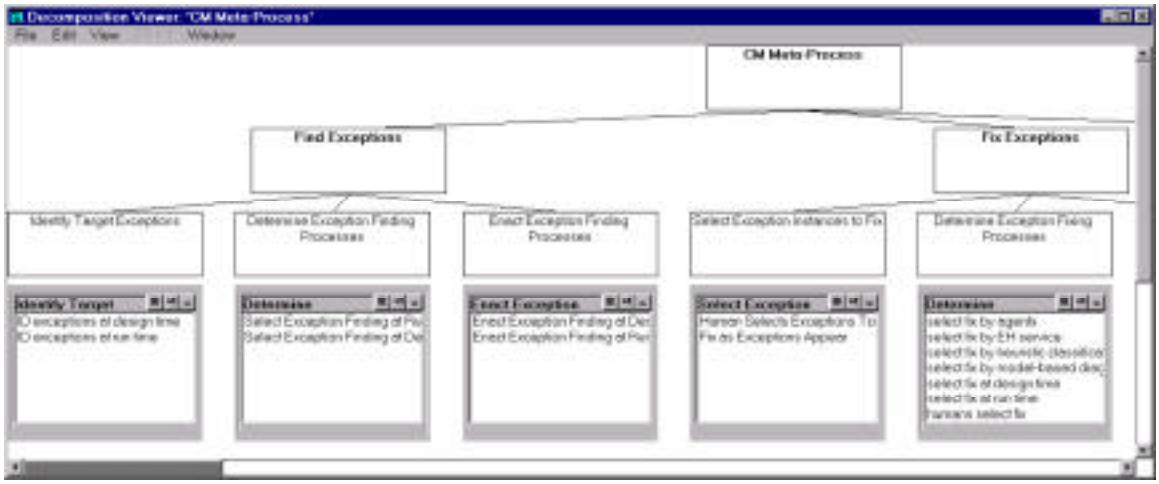


Figure 7. Snapshot of the process recombinator.

The Recombinator allows one to select, for each subtask, the alternatives that one is interested in, and automatically generates all the combinations thereby implied. It is up the human user, however, to determine which of the candidate exception management meta-processes so generated are most appropriate for their particular context.

Facilitating Research: An exception repository can serve as a valuable resource for fostering more effective, accumulative and cross-disciplinary research on exception management, in several important ways. The cross-disciplinary transfer of ideas is facilitated because exception management expertise is captured in a single repository indexed by functional similarity rather than source domain. The taxonomic structure of the repository facilitates *finding gaps* in the exception management knowledge. One can, for example, look for exception types with no associated exception handlers. The Web version of the exception repository also supports *structured discussions* by allowing one to create discussion forums organized around focus topics such as filling in a particular branch of a taxonomy, adding to a tradeoff table, or detailing a particular process description. It has been our experience that such foci can be more effective than unstructured discussions for capturing process knowledge.

Evaluation of The Contributions of This Work

We have been unable to uncover any previous effort to produce a systematic multi-disciplinary repository of exception handling expertise applicable to multi-agent systems. There have been a few analogous efforts to capture knowledge about handling the subset of exceptions known as multi-agent conflicts (Matta 1996) (Castelfranchi 1996) (Ramesh

and Sengupta 1994) (Feldman 1985) but they leave out other exception types, and are lacking key concepts incorporated in the MIT exception repository, such as the coordination mechanism to exception mapping, the exception management meta-process, and the specialization hierarchy of exception handling processes.

The MIT exception repository has been evaluated only on a limited internal basis to date, so it is premature to draw definitive conclusions about its utility for students, researchers and practitioners. Our experience to date suggests, however, that the repository is likely to have significant value. The repository currently includes knowledge derived from roughly 60 research publications, in addition to our own analyses of the exception types and associated handlers for the contract net (Smith and Davis 1978) (Aryananda 1999), multi-level coordination (Durfee and Montgomery 1990), and TEAMCORE (Tambe M 1997) (Xu 1999) protocols, representing instances of the most widely-used and studied classes of multi-agent coordination mechanisms. It is our judgement that the schema presented above captures all the significant aspects of the exception management information we have encountered to date. It is also clear, in addition, that the Process Handbook provides a high level of enabling technology for repository purposes, including a growing set of sophisticated search, navigation, business process re-design and structured discussion tools. Previous experience with these tools suggests that they can be powerful enablers. The Handbook has been successfully used, for example, to teach classes at the Sloan School of Management as well as Babson College. The Handbook process redesign methodology has been applied in several domains, most recently (in cooperation with the consulting firm AT Kearney) to re-design the hiring processes in a major financial services firm. The participants in this study felt that the approach was effective in generating a much wider range of novel and promising process alternatives than would have been uncovered by traditional methods (Herman, Klein et al. 1998).

Future Work

The MIT exception repository is a work in progress. We will continue to add to the repository content, drawing from multiple disciplines. We plan to explore the use of additional repository structuring schemes and tools, such as the notion of a “guided tour’ that provides a suggested sequence for traversing the repository links for specific pedagogical purposes. We are also evolving the repository into the basis of a domain-independent exception handling service that can be ‘plugged’ in to existing multi-agent systems (Klein and Dellarocas 1999). This has involved augmenting the repository schema to capture causal links between different exception types to allow heuristic classification of failure modes. Our hope is to evolve the exception repository into a self-sustaining Web-based community resource, which will require addressing technological

issues (for example, developing a Web-based authoring tool) as well as sociological issues (particularly concerning incentives for adding content).

For additional information about this and related work, including eventual access to the MIT exception repository itself, please see http://ccs.mit.edu/klein/.

Acknowledgements

This work was supported by NSF grant IIS-9803251 (Computation and Social Systems Program) and by DARPA grant F30602-98-2-0099 (Control of Agent Based Systems Program). I am grateful for many helpful comments from the members of the MIT Center for Coordination Science: John Quimby, George Herman, George Wyner, Abraham Bernstein, and Prof. Thomas Malone.

References

Adamides, E. and D. Bonvin (1993). "Failure recovery of flexible production systems through cooperation of distributed agents." Ifip Transactions B: Computer Applications in Technology **11**: 227-38.

Aryananda, L. (1999). An Exception Handling Service for the Contract Net Protocol. Department of Electrical Engineering and Computer Science. Cambridge MA, MIT.

Auramaki, E. and M. Leppanen (1989). Exceptions and office information systems. Proceedings of the IFIP WG 8.4 Working Conference on Office Information Systems: The Design Process., Linz, Austria.

Bansal, A. K., K. Ramohanarao, et al. (1997). Distributed Storage of Replicated Beliefs to Facilitate Recovery of Distributed Intelligent Agents. Intelligent Agents IV; Proceedings of ATAL-97. M. P. Singh, A. Rao and M. J. Wooldridge**:** 77-91.

Bernstein, A., M. Klein, et al. (1999). The Process Recombinator: A Tool for Generating New Business Process Ideas. Proceedings of the International Conference on Information Systems, Charlotte, North Carolina USA.

Birnbaum, L., G. Collins, et al. (1990). Model-Based Diagnosis of Planning Failures. Proceedings of the National Conference on Artificial Intelligence (AAAI-90).

Broverman, C. A. and W. B. Croft (1987). Reasoning About Exceptions During Plan Execution Monitoring. Proceedings of the National Conference on Artificial Intelligence (AAAI-87).

Burns, A. and A. Wellings (1996). Real-Time Systems and Their Programming Languages, Addison-Wesley.

Castelfranchi, C. (1996). Conflict Ontology. Proceedings of the Workshop on Conflict Management, European Conference on Artificial Intelligence (ECAI).

Chia, M. H., D. E. Neiman, et al. (1998). Poaching and distraction in asynchronous agent activities. Proceedings of the Third International Conference on Multi-Agent Systems, Paris, France.

Chiu, D. K. W., K. Karlapalem, et al. (1997). Exception Handling in ADOME Workflow System. Hong Kong, Hong Kong University of Science and Technology.

Decker, K., K. Sycara, et al. (1997). Middle-agents for the Internet. Proceedings of IJCAI-97, Nagoya, Japan.

Dellarocas, C. and M. Klein (1999). Towards civil agent societies: Creating robust, open electronic marketplaces of contract net agents. Proceedings of the International Conference on Information Systems (ICIS-99), Charlotte, North Carolina USA.

Durfee, E. H. and T. A. Montgomery (1990). A Hierarchical Protocol for Coordinating Multiagent Behaviors.

Feldman, D. C. (1985). "A Taxonomy Of Intergroup Conflict Resolution Strategies." The 1985 Annual Conference on Developing Human Resources.

Finin, T., Y. Labrou, et al. (1997). KQML as an agent communication language. Software Agents. J. Bradshaw. Cambridge MA, MIT Press.

Finkelstein, A., D. Gabbay, et al. (1994). "Inconsistency Handling in Multi-perspective Systems." IEEE Transactions on Software Engineering **20**(8): 569-578.

Firby, R. J. (1987). An Investigation into Reactive Planning in Complex Domains. Proceedings of AAAI-87.

Fletcher, M. and D. S. Misbah (1999). "Task rescheduling in multi-agent manufacturing." Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA **99**: 689-94.

Hägg, S. (1996). A Sentinel Approach to Fault Handling in Multi-Agent Systems. Proceedings of the Second Australian Workshop on Distributed AI, in conjunction with

Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96), Cairns, Australia.

Herman, G., M. Klein, et al. (1998). A Template-Based Process Redesign Methodology Based on the Process Handbook. Unpublished discussion paper. Cambridge MA, Center for Coordination Science, Sloan School of Management, Massachussetts Institute of Technology.

Hindriks, K., F. de Boer, et al. (1998). "Failure, monitoring and recovery in the agent language 3APL." Cognitive Robotics. Papers from the.

Howe, A. E. (1995). "Improving the reliability of artificial intelligence planning systems by analyzing their failure recovery." IEEE Transactions on Knowledge and Data Engineering **7**(1): 14-25.

Kaminka, G. and M. Tambe (1997). "Towards social comparison for failure detection." Socially Intelligent Agents. Papers from the.

Kaminka, G. A. and M. Tambe (1997). Social Comparison for Failure Detection and Recovery. Intelligent Agents IV; Proceedings of ATAL-97. M. P. Singh, A. Rao and M. J. Wooldridge**:** 127.

Katz, D. M., S. (1993). Exception management on a shop floor using online simulation. Proceedings of 1993 Winter Simulation Conference - (WSC '93), Los Angeles, CA, USA, IEEE; New York, NY, USA.

Klein, M. (1998). A Knowledge-Based Approach to Handling Exceptions in Workflow Systems. Cambridge MA USA, MIT Center for Coordination Science.

Klein, M. and C. Dellarocas (1999). Exception Handling in Agent Systems. Proceedings of the Third International Conference on AUTONOMOUS AGENTS (Agents '99), Seattle, Washington.

Malone, T. W., K. Crowston, et al. (1998). "Tools for inventing organizations: Toward a handbook of organizational processes." Management Science **45**(3): 425-443.

Malone, T. W. and K. G. Crowston (1994). "The interdisciplinary study of Coordination." *ACM Computing Surveys* **26**(1): 87-119.

Matta, N. (1996). Conflict Management in Concurrent Engineering: Modelling Guides. ECAI Workshop on Conflict Management.

Mi, P. and W. Scacchi (1993). Articulation: An Integrated Approach to the Diagnosis, Replanning and Rescheduling of Software Process Failures. Proceedings of 8th Knowledge-Based Software Engineering Conference, Chicago, IL, USA, IEEE Comput. Soc. Press; Los Alamitos, CA, USA.

Mullender, S. J. (1993). Distributed systems. New York

Ramesh, B. and K. Sengupta (1994). "Managing Cognitive and Mixed-Motive Conflicts in Concurrent Engineering." Concurrent Engineering Research and Applications: Special Issue on Conflict Management in Concurrent Engineering **II**(3): 223-236.

Sandholm, T., S. Sikka, et al. (1999). Algorithms for Optimizing Leveled Commitment Contracts. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden.

Smith, R. G. and R. Davis (1978). "Distributed Problem Solving: The Contract Net Approach." Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence.

Tambe M (1997). "Towards flexible teamwork." Journal of Artificial Intelligence Research **7**: 83-124.

Traverso, P., L. Spalazzi, et al. (1996). "Reasoning about acting, sensing and failure handling: a logic for agents embedded in the real world." Intelligent Agents II. Agent Theories, Architectures, and Languages. IJCAI`95 Workshop.

Venkatraman, M. and M. P. Singh (1999). "Verifying Compliance with Commitment Protocols: Enabling Open Web-Based Multiagent Systems." Autonomous Agents and Multi-Agent Systems **3**(3).

Xu, W. (1999). Generic Exception Analysis in a Dynamic Multi-Agent Environment. Department of Electrical Engineering and Computer Science. Cambridge MA, MIT.

Appendix C: A Knowledge-Based Methodology for Designing Reliable Multi-Agent Systems

Mark Klein
Massachusetts Institute of Technology
Cambridge MA 02139
(617) 253-6796
m_klein@mit.edu

ABSTRACT

This paper describes a methodology that system designers can use to identify, and find suitable responses for, potential failure modes (henceforth called 'exceptions') in multi-agent systems.

1. INTRODUCTION

Multi-agent systems must be able to operate robustly despite many possible failure modes ('exceptions') that can occur. Traditionally, multi-agent system (MAS) designers have largely relied on their experience and intuition in order to anticipate all the ways their systems can fail, and how these problems can best be addressed. While methodologies such as failure mode effects analysis (FMEA) do exist [1], they simply provide a systematic procedure for analyzing systems, without offering specific insights into what exceptions can occur or how they can be resolved.

This approach is becoming untenable, however, as the scale, heterogeneity and openness of multi-agent systems increases. Multi-agent systems, with their promise of self-organized behavior, are being looked to as a way to smoothly and rapidly integrate the activities of large collections of software entities that may never have worked together before. The agents in such 'open' contexts will not have been designed under centralized control, and must operate on the infrastructures at hand. Such systems must be able to operate effectively despite a bewildering range of possible exceptions. We have identified two main classes of exceptions that can occur in MAS contexts:

- Commitment Violations: This category consists of problems where some entities in the MAS do not properly discharge their commitments to each other, e.g. when a subcontractor is overdue with a task, a message is delivered garbled or late, or a host computer crashes. Even the best production code includes an average of 3 design

faults per 1000 lines of code [2], and in open systems we can expect a wide range of code quality as well as actively malicious agents.

- Emergent Dysfunctions: This category consist of dysfunctional behaviors that emerge from the locally correct behavior of many agents. There are many examples of such dysfunctions, ranging from social dilemmas such as the 'tragedy of the commons' [3], to wild variations in resource utilization [4] [5], and timing artifacts such as 'resource poaching' (wherein earlier low priority tasks freeze out later high-priority tasks from access to critical resources) [6]. Such exceptions are especially problematic because they do not represent errors per se, but rather the unexpected consequences of simple coordination mechanisms applied in complex environments.

The challenge of identifying exceptions and their resolutions is complicated by the fact that expertise on this subject is scattered across multiple disciplines that include computer science, industrial engineering, economics, management science, biology, and the complex system sciences. MAS designers are thus unlikely to be cognizant of all the expertise potentially relevant to their tasks.

This paper describes a methodology that multi-agent system (MAS) designers can use to identify, and find suitable responses for, these potential failures (henceforth called 'exceptions'). We present the core exception analysis methodology in section 2, and then describe (in section 3) how an augmentation of the MIT Process Handbook captures exception handling expertise in a way that can greatly increase the speed and comprehensiveness of exception analysis.

2. EXCEPTION ANALYSIS

Our exception analysis methodology is based on the insight that coordination fundamentally involves the making of commitments [7] [8] [9], and that exceptions (i.e. coordination failures) can as a result be viewed as *violations* of the commitments agents require of one another. Exception analysis thus consists of the following steps:

- Identify the *commitments* agents require of one another
- Identify the *processes* by which these commitments are achieved
- Identify the ways these processes can violate these commitments (i.e. the *exceptions*)
- Identify the ways these exception can be handled (i.e. the exception *handlers*)

We consider each of these steps in the paragraphs below. To make the discussion more concrete, we will describe them in context of the well-known auction-based task-sharing mechanism known as the "Contract Net" (CNET) [10].

In this protocol, an agent (the "contractor") identifies a task that it cannot or chooses not to do locally and attempts to find another agent (the "subcontractor") to perform the task. It begins by creating a Request For Bids (RFB) which describes the desired work, and then sending it to potential subcontractors (typically identified using an agent known as a 'matchmaker'). Interested subcontractors respond with bids (specifying such issues as the price and time needed to perform the task) from which the contractor selects a winner. This is thus a first-price sealed-bid auction. The winning agent, once notified of the award, performs the work (potentially subcontracting out its own subtasks as needed) and submits the results to the contractor.

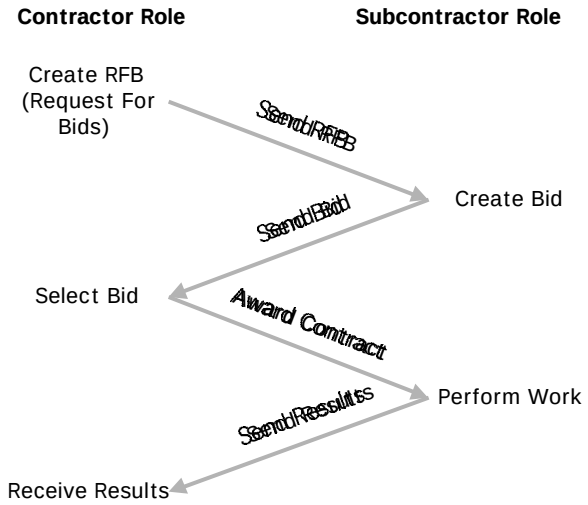


Figure 1. The Contract Net Coordination Mechanism.

We can see that there are thus at least three key agent types in CNET: the contractor, subcontractor, and matchmaker.

Identifying Commitments & Processes: The commitments involved in a coordination mechanism represent all the places where one agent depends on some other agent to achieve its goals [7]. They can be identified in a straightforward way by linking commitments to the agent processes that achieve them, and linking these processes to the commitments they in turn require to execute successfully. The first commitment in CNET, for example, is the contractors' requirement that it have a list of agents potentially suitable for performing the task T it wishes to subcontract out:

C1:*matchmaker* receives list of subcontractor agents suitable for task T

In CNET, this requirement is discharged by the matchmaking process enacted by the matchmaker agent:

P1: *matchmaker* finds matches for skill set S

In order for process P1 to discharge commitment C1, it in turn requires such other commitments as:

C2: *matchmaker* receives subcontractor agent skill notifications
C3: *matchmaker* receives correct skill set S in timely way
C4: *matchmaker* receives sufficient computational resources to run effectively
C5: *matchmaker* code was programmed correctly
C6: *matchmaker* results message is sent quickly and correctly to requesting contractor agent

Commitment C4, in turn, is achieved by a host computer:

P2: *host computer* provides computational resources for hosted agents

If we follow this process in sufficient depth we can, in principle, exhaustively identify the commitments (and associated processes) involved in a given coordination mechanism.

The explicit identification of all commitments is critical because often exceptions occur because no mechanism was put in place to ensure the satisfaction of some important but overlooked implicit commitment. Also note that commitments should be enumerated *from the idealized perspective of each agent.* For example, in an auction the seller ideally wants the following commitment:

C6: *seller* receives bids representing true bidder valuation for good

even though in many situations the bidder may have no intention of fulfilling this commitment (e.g. in auctions where true-value revelation is not the dominant strategy). Many important exceptions represent violations of such ideal-case commitments.

Identifying Commitment Violations (Exceptions): The next step is to identify, for each commitment, how that commitment can be violated by the process selected to achieve it, i.e. what the possible exceptions are. An initial set of exceptions can be identified simply as the logical negations of the commitment itself. For example, the commitment to send a message consists of three components: delivering the *right message* to the *right place* at

the *right time*. Any process selected to achieve these commitments thus has three possible failure modes:

E1: *sender* delivers the wrong message (e.g. the message is garbled)
E2: *sender* delivers message to the wrong place
E3: *sender* delivers message at the wrong time (e.g. the message arrives late or never)

Not all exception types, however, can be identified so simply. Process P2 above, for example, can violate its commitments due to exceptions that include:

E4: *host computer* experiences a denial of service attack
E5: *host computer* is infected by a virus

The range of possible exception types seems to be limited only by human imagination. This introduces a experiential component into exception analysis; one must rely on one's previous experience to identify possible exception types, and an exhaustive identification may not be possible

Identifying Exception Handlers: Once we have identified the exceptions that potentially characterize a given MAS process, we are ready to identify possible processes for *handling* these exceptions. Exception E1, for example, can be *detected* by the process:

P3: *sender* performs error-detecting checksum on message contents

and it can be *resolved* by:

P4: *sender* re-sends message

As with exceptions themselves, the range of possible exception handling processes appears to be limited only by human creativity. Also note that exception handling processes, just like any other MAS process, can of course require their own commitments and face their own exceptions.

This exception analysis procedure is systematic but potentially very time-consuming, and it still requires that MAS designers have a substantial amount of expertise about possible exceptions and how they can be handled, so the possibility of missing important exceptions or valuable exception handling techniques remains.

3. EXPLOITING A KNOWLEDGE BASE

This challenge has led us to explore whether it is possible to systematically accumulate exception-related expertise so that designers can benefit from ideas drawn from other designers, and from multiple disciplines, in order to perform exception analysis more quickly and completely. Our approach has been to build upon the MIT Process Handbook, a process knowledge repository which has been under development at the Center for Coordination Science (CCS) for about 10 years [11]. The key concept underlying the Handbook is that processes can be arranged into a taxonomy, with very generic processes at one extreme and increasingly *specialized* processes towards the other. Such taxonomies have two useful properties. One is that attributes of generic entities tend to be inherited by their 'specializations', so one can capture useful generalizations that apply to a wide range of processes. The other is that similar entities (e.g. processes with similar purposes) tend to appear close to one another.

We extended this schema to allow it to capture the results of applying the exception analysis methodology described above. This was accomplished (see [12] [13]) by defining:

- a taxonomy of commitment types, where commitments can be linked to the processes that require them as well as the processes that achieve them, and
- a taxonomy of exception types, where exceptions can be linked to the processes they impact as well as the processes appropriate for handling them.

Using this extended schema, we have developed a knowledge base that consists of the results of applying our exception analysis methodology to a range of more or less abstract MAS coordination processes and their component sub-processes. We have also implemented a web-based interface for accessing and editing the contents of this knowledge base. The examples presented in this paper are all drawn from this knowledge base.

A MAS designer can use such a knowledge base to facilitate exception analysis as follows:

- Consult the knowledge base to find the generic processes that subsume, or closely match, the processes used in the MAS of interest.
- Identify which of the exceptions listed for those generic processes in the knowledge base appear to be important for this particular MAS.
- For each of these exceptions, identify which of the exception handler(s) described in the knowledge base seem best suited for this MAS. These exception handlers should,

of course, be submitted to the same exception analysis procedures as the other MAS processes.

The power of this approach comes from the fact that a relatively small corpus of abstract commitments, exceptions and process models is, when represented in this way, capable of capturing a surprisingly high proportion of the exception handling expertise we need. We describe how this works in more detail below.

Finding the matching generic processes: The taxonomic organization of processes in the Handbook knowledge base makes it straightforward to find matching generic process(es). The procedure is similar to finding a book in a library. One simply traverses down the subject taxonomy from the top, selecting the most appropriate sub-categories at each step, until the desired section is reached.

We have developed a taxonomy of the most widely-used MAS coordination processes, ranging from market mechanisms to distributed planning to game-theoretic and stigmergic approaches. We have focused our initial efforts on auctions because their wide applicability, scalability, simplicity and well-understood properties make them widely used by MAS designers. Every auction mechanism captured in the Handbook includes a textual description as well as an enumeration of its subtasks, required commitments, and the commitments it achieves.

Using this information, one can readily determine which processes in the taxonomy match the process of interest. Imagine for example that a MAS designer has developed a task allocation scheme wherein subcontractor agents send in sealed bids in order to compete for subtasks. To find a matching generic process, he or she would start at the 'resource sharing process' branch of the taxonomy, and traverse down from there, quickly reaching the process taxonomy branch devoted to auction mechanisms:

Figure 2. A portion of the auction mechanism taxonomy.

Using this portion of the taxonomy, the designer can quickly determine that his or her task allocation scheme is an instance of the generic Contract Net process described in the knowledge base. In addition to enabling the quick identification of relevant exception types (see below), finding the matching generic processes can increase the completeness of the MAS model: one can look at the components and requirements in the generic process and check whether any are missing in the current MAS model.

Finding Applicable Exceptions: Once the matching generic processes have been identified, we can then identify the exceptions that the MAS is potentially prone to. This is straightforward because each process in the knowledge base is linked directly to its characteristic exceptions. All auctions, for example, are a specialization of the abstract 'pull-based sharing' process, which represents mechanisms wherein resources, e.g. subcontractor agents, are allocated based on consumer requests rather centralized budgeting. If we consult the Handbook knowledge base we find that pull-based sharing is prone to such 'emergent dysfunction' exceptions as "resource poaching" and "synchronized jump thrashing":

Figure 3. An example of exceptions for a generic process.

The fact that auction mechanisms are prone to such emergent dysfunctions is a potentially powerful and easily missed insight. Auction mechanisms are typically designed by economists interested in equilibrium behavior, and implemented by computer scientists, while the dynamics of resource sharing are studied by researchers in the complex systems field, which grew mainly out of physics. This is an example, therefore, of how a taxonomic approach, based as it is on the identification of powerful generalizations, can foster the cross-disciplinary transfer of insights about exception handling challenges and solutions.

Finding Applicable Handlers: Once one has determined which exceptions are important for a particular MAS, the next step is to identify the appropriate exception handlers. This is straightforward because exceptions in the Handbook knowledge base are linked directly to the exception handling processes appropriate for them. Our current knowledge base, for example, notes that the exception "Synchronized Jump Thrashing" (where resource consumers generate oscillatory or even chaotic resource utilization behavior due to delayed resource quality information) can be detected using signal processing techniques and resolved by carefully timed modification of resource availability messages [4].

There are four classes of exception handlers in the knowledge base, divided into two pairs. If a exception has not yet occurred, we can use:

[1] Exception *anticipation* processes, which uncover situations where a given class of exception is likely to occur**.** Resource poaching, for example, can be anticipated when there is a flood of long duration tasks requiring scarce, non-preempting subcontractors to perform them.

- Exception *avoidance* processes, which reduce or eliminate the likelihood of a given class of exception. Resource poaching can be avoided, for example, by allowing subcontractors to preempt their current tasks in favor of higher priority pending tasks.

If the exception has already occurred, we can use:

- Exception *detection* processes, which detect when an exception has actually occurred. Some exceptions, such as bidder collusion for example, are difficult to anticipate but can be detected post-hoc by looking at bid price patterns.
- Exception *resolution* processes, which resolve an exception once it has happened. One resolution for bidder collusion, for example, is to penalize and/or kick out the colluding bidders and re-start the auction for the good in question.

The exceptions in our knowledge base are arranged, like processes, into a taxonomic structure:

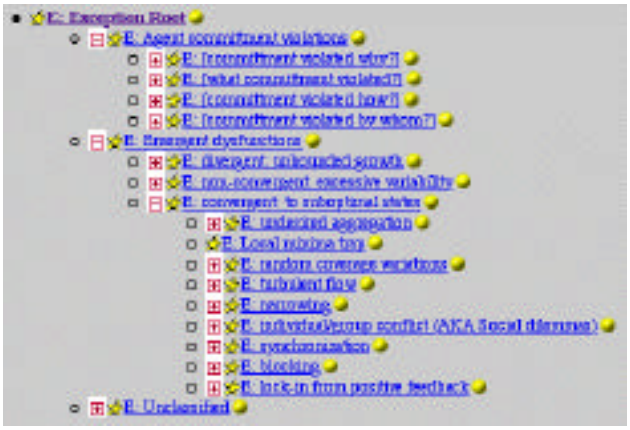


Figure 4. A fragment of the exception taxonomy.

Exceptions are grouped into classes that share similar underlying causes and thus similar exception handling techniques. This implies that when a new exception handling process is entered into the knowledge base, it can be placed in a way that makes explicit the full range of exceptions to which it is applicable.

Where multiple alternative handler processes exist for addressing a particular exception, the Handbook knowledge base allows one to describe the pros and cons of the handlers using tradeoff tables. The taxonomic structure of the knowledge base also facilitates the design of innovative handler processes through re-combining elements of existing handlers [14].

Whatever handlers we select can themselves be made subject to the exception analysis approach described above in order to further increase the robustness of the MAS. Reputation mechanisms, for example, have been put forth as handlers for many classes of agent commitment violation exceptions. Our knowledge base captures the fact that such mechanisms can be sabotaged by such exceptions as dishonest ratings.

4. CONTRIBUTIONS

Limited space has only allowed us to sketch out the knowledge-based exception analysis approach we have been developing. We hope, however, that the key benefits have been made clear. Existing exception analysis techniques leave the identification of possible exceptions and handlers up to the MAS designers themselves. The knowledge-based exception analysis procedure we describe, by contrast, makes it possible to systematically enumerate all the points where exceptions can occur, quickly identify what exceptions may appear at those points, and suggest how they may be handled. Because this knowledge base represents the accumulation of expertise drawn from many different

designers and disciplines, it has the potential of identifying exception types and handler techniques that may otherwise be overlooked by the MAS designer.

We have applied our tools and knowledge base to exception analysis in a range of domains including futures trading, multi-agent system task allocation, and concurrent engineering. Our preliminary assessment is that the methodology can be effective in helping designers design more reliable multi-agent systems.

5. FUTURE WORK

We are continuing to accumulate a knowledge base of exception types and handlers, currently focusing on market-based sharing, collaborative synthesis, and emergent dynamical dysfunctions. We are also developing software agents that use this knowledge base to do real-time exception detection and intervention in multi-agent systems. For additional information about this and related work, see http://ccs.mit.edu/klein/.

6. ACKNOWLDGEMENTS

This work was supported by the DARPA CoABS program as well as the NSF Computational and Social Systems program. The author gratefully acknowledges the important contributions made by Chrysanthos Dellarocas, George Herman, Thomas Malone, Simon Parsons, John Quimby, Juan-Antonio Rodriguez-Aguilar, and others.

REFERENCES

[1] Hunt, J., D.R. Pugh, and C.J. Price. *Failure mode effects analysis: a practical application of functional modeling.* Applied Artificial Intelligence, 1995. 9(1): p. 33-44.

[2] Gray, J. and A. Reuter. *Transaction Processing : Concepts and Techniques*. Morgan Kaufmann series in data management systems. 1993, San Mateo, Calif. USA: Morgan Kaufmann Publishers. xxxii, 1070.

[3] Hardin, G.,*The Tragedy of the Commons.* Science, 1968. 162: p. 1243 - 1248.

[4] Huberman, B.A. and D. Helbing. *Economics-based optimization of unstable flows.* Europhysics Letters, 1999. 47(2): p. 196-202.

[5] Sterman, J.D. *Learning in and about complex systems*. 1994, Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts Institute of Technology. 51.

[6] Chia, M.H., D.E. Neiman, and V.R. Lesser. *Poaching and distraction in asynchronous agent activities*. In *Proceedings of the Third International Conference on Multi-Agent Systems*. 1998. Paris, France.

[7] Singh, M.P. *An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts.* Artificial Intelligence and Law, 1999. 7: p. 97-113.

[8] Jennings, N.R. *Coordination Techniques for Distributed Artificial Intelligence*, in *Foundations of Distributed Artificial Intelligence*, G.M.P. O'Hare and N.R. Jennings, Editors. 1996, John Wiley & Sons. p. 187-210.

[9] Gasser, L. *DAI Approaches to Coordination*, in *Distributed Artificial Intelligence: Theory and Praxis*, N.M. Avouris and L. Gasser, Editors. 1992, Kluwer Academic Publishers. p. 31-51.

[10] Smith, R.G. and R. Davis. *Distributed Problem Solving: The Contract Net Approach.* Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence, 1978.

[11] Malone, T.W., et al. *Tools for inventing organizations: Toward a handbook of organizational processes.* Management Science, 1999. 45(3): p. 425-443.

[12] Klein, M. and C. Dellarocas. *A Knowledge-Based Approach to Handling Exceptions in Workflow Systems.* Journal of Computer-Supported Collaborative Work. Special Issue on Adaptive Workflow Systems., 2000. 9(3/4).

[13] Klein, M. *Towards a Systematic Repository of Knowledge About Managing Collaborative Design Conflicts*. in *Proceedings of the International Conference on AI in Design (AID-2000)*. 2000. Boston MA: Kluwer Academic Publishers.

[14] Bernstein, A., M. Klein, and T.W. Malone. *The Process Recombinator: A Tool for Generating New Business Process Ideas*. in *Proceedings of the International Conference on Information Systems (ICIS-99)*. 1999. Charlotte, North Carolina USA.

Appendix D: Handling Resource Use Oscillation in Open Multi-Agent Systems

Mark Klein
Massachusetts Institute of Technology
NE20-336
Cambridge MA 02132
Tel: (617) 253-6796
Fax: (617) 452-3231
m_klein@mit.edu

Yaneer Bar-Yam
New England Complex Systems Institute
24 Mt. Auburn Street
Cambridge, MA 02138
Tel: (617) 547-4100
Fax: (617) 661-7711
yaneer@necsi.org

The Challenge

The convergence of ubiquitous electronic communications such as the Internet, electronic agents acting as proxies for human consumers, and web/grid service standards such as XML are rapidly ushering in a world where hordes of software agents, acting for humans, can rapidly select among multitudes of competing providers offering almost every imaginable service. This is inherently an "open" world, a marketplace where the agents operate as peers, neither designed nor operated under central control. Such a world offers the potential for unprecedented speed and efficiency in getting work done.

In such open peer-to-peer systems we face, however, the potential of highly dysfunctional dynamics emerging as the result of many locally reasonable agent decisions [1]. Such "emergent dysfunctions" can take many forms, ranging from inefficient resource allocation [2] to chaotic inventory fluctuations [3] [4] to non-convergent collective decision processes [5]. This problem is exacerbated by the fact that agent societies operate in a realm whose communication and computational costs and capabilities are radically different from those in human society, leading to collective behaviors with which we may have little previous experience. It has been argued, for example, that the 1987 stock crash was due in part to the action of computer-based "program traders" that were able to execute trade decisions at unprecedented speed and volume, leading to unprecedented stock market volatility [6].

Let us focus on one specific example of emergent dysfunctional behavior: resource use oscillation in request-based resource sharing. Imagine that we have a collection of consumer agents faced with a range of competing providers for a given resource (e.g. a piece of information such as a weather report, a sensor or effector, a communication link, a storage or computational capability, or some kind of data analysis). Typically, though not exclusively, the utility offered by a resource is inversely related to how many consumers are using it. Each agent strives to select the resource with the highest utility (e.g. response time or quality), and resources are allocated first-come first-served to those

who request them. This is a peer-to-peer mechanism: there is no one ‘in charge’. This kind of resource allocation is widely used in settings that include fixed-price markets, internet routing, and so on. It is simple to implement, makes minimal bandwidth requirements, and - in the absence of delays in resource status information – allows consumers to quickly converge to a near optimal distribution across resources (see figure 1 below).

Consumers, however, will often have a somewhat delayed picture of how busy each resource is. Agents could imaginably poll every resource before every request. This would cause, however, a N-fold increase in number of required messages for N servers, and does not eliminate the delays caused by the travel time for status messages. In a realistic open system context, moreover, consumers probably cannot fully rely on resource providers to accurately characterize the utility of their own offerings (in a way that is comparable, moreover, across providers). Resource providers may be self-interested and thus reluctant to release utilization information for fear of compromising their competitive advantage. In that case, agents will need to estimate resource utilization using other criteria such as their own previous experience, consulting reputation services, or watching what other consumers are doing. Such estimates are almost certain to lag at times behind the actual resource utility.

When status information is delayed in some way, we find that resource use oscillations emerge, potentially reducing the utility achieved by the consumer agents far below the optimal value predicted by an equilibrium analysis [7].What happens is the following. Imagine for simplicity that we have just two resources, R1 and R2. We can expect that at some point one of the resources, say R1, will be utilized less than the other due to the ebb and flow of demand. Consumer agents at that point will of course tend to select R1. The problem is that, since their image of resource utilization is delayed, they will continue to select R1 even after it is no longer the less utilized resource, leading to an “overshoot” in R1’s utilization. When the agents finally realize that R2 is now the better choice, they will tend to select R2 with the same delay-induced overshoot. The net result is that the utilization of R1 and R2 will oscillate around the optimal equilibrium value. The range of the oscillations, moreover, increases with the delay, to the extent that all the consumers may at times select one server when the other is idle:

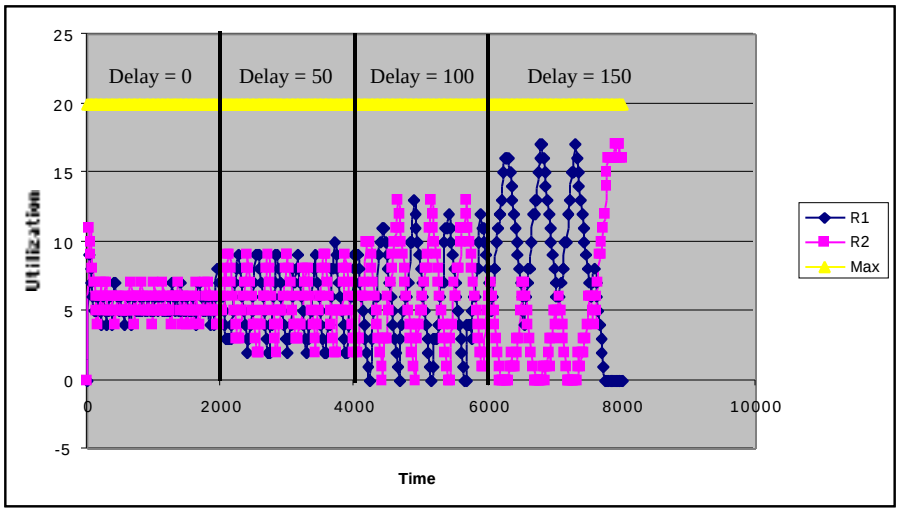


Figure 1. Utilization of two equivalent resources with and without delays in status information.

Oscillations have two undesirable effects. One is that they can reduce the utility received by consumers below optimal values. The other is that they can increase the variability of the utility achieved by the consumers, which may be significant in domains where consistency is important. The precise impact of oscillations is driven by the relationship between resource utilization and utility. Let us consider several scenarios.

Imagine we have grocery store customers (consumers) choosing from two checkout lines (resources). Their utility is given by how long they have to wait. The time needed to check out each customer is not affected by the length of a line, so the wait at each line is *negatively* and *linearly* impacted by its length:

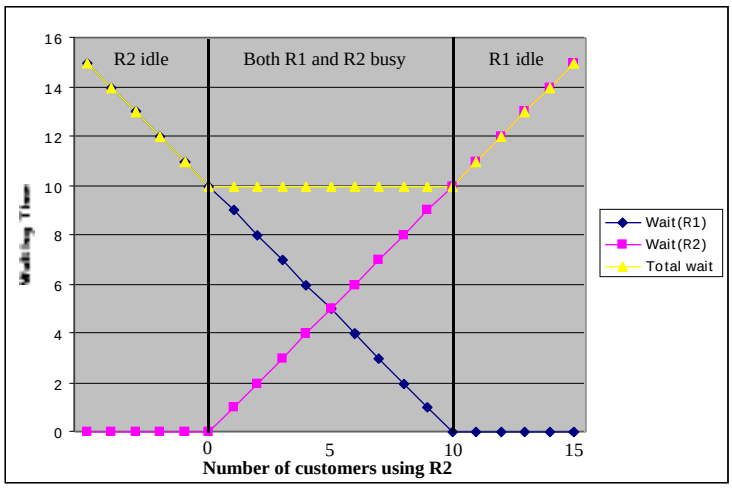


Figure 2: Average grocery store checkout times as a function of distribution across lines.

When both checkout lines are busy, oscillations in the distribution of customers across lines do not affect average check out times. The longer waits faced by customers in one line are exactly offset by the shorter waits enjoyed by the customers in the other line. The *variance* of checkout times will, of course, be wider if there are oscillations. Note, also, that if the oscillations are so severe that one line goes idle while the other has multiple customers, the average waiting times will increase because only one resource is active.

Now consider, by contrast, movie goers faced with a choice of theatres to attend. Each movie goer wants the best seat possible. The fuller the theatre, the poorer the seats that remain, so the utility of the seats is *negatively* and *non-linearly* impacted by the number of people using the theatre:

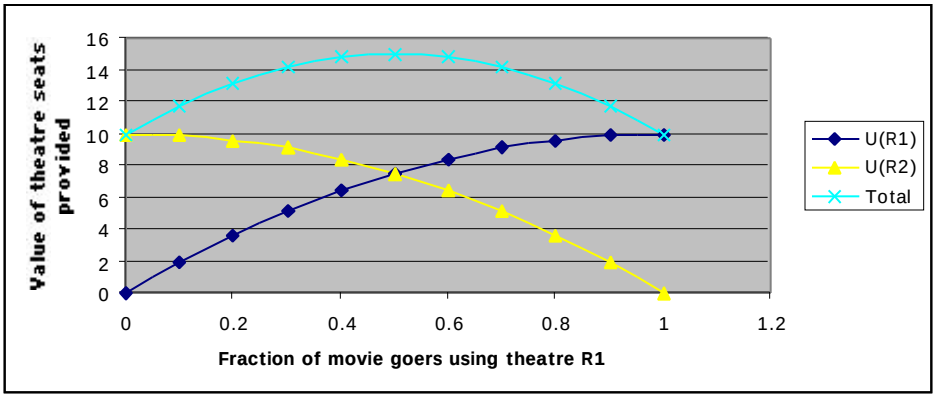


Figure 3: The value of movie seats as a function of distribution across theatres[2].

There is one optimal distribution of movie goers across theatres (50-50 in this case), and oscillations in the actual distribution reduce the average utility because much of the time is spent at a non-optimal distribution. Oscillations increase the variance in this case as well.

A third scenario involves TV viewers (the consumers) with a choice of shows (the resources) to watch. The utility of a show for a given consumer is *not* impacted by who else happens to be watching it, so it is a constant function of the utilization. In this case we would expect no persistent oscillations to occur, because changing viewership does not impact the movement of viewers among shows. Any variations that do occur would, in any case, have no impact on the utility received by the consumers.

The last scenario involves club goers (consumers) with a choice of night clubs (resources) they can attend. The value of a club to them is given by how many other

[2] For this example, the utility of the seats provided by a theatre was given by $U(f) = 20*f - 10*f^2$, where f is the fraction of moviegoers who selected that theatre.

people are there. The resource utility is thus *positively* impacted by the number of consumers. In this scenario, there will be no persistent oscillations because whatever club achieves an initial advantage in number of participants will eventually get all of the club goers.

We can therefore distinguish four scenarios for request-based resource sharing. All are potentially important in open agent system contexts. Network routers operate using the "Grocery Checkout" model. A broadcast operates using the "TV show" model. Auctions arguably fit into the "Nightclub" model. And providers of varying utility resources such as sensors, effectors, or storage capacity fit into the "Movie Theatre" model. These scenarios vary however in their susceptibility to the emergent dysfunction of resource use oscillation. The key factor concerns how the utility offered by a resource varies with its utilization:

	Linear Relationship	**Non-linear Relationship**
Positive Impact	Nightclub	
Zero Impact	TV Show	
Negative Impact	Grocery Checkout	Movie Theatre

Table 1. The cases for how resource utility is affected by utilization.

In two of these cases, where resource utility is a constant or positive function of resource utilization, resource use oscillation will not occur and/or have no impact. In the remaining scenarios, however, oscillations can occur and negatively impact the consistency and/or average value of the utility achieved by consumers. We will confine our attention to these latter cases in the remainder of the paper.

Previous Work

What can be done about resource use oscillation in request-based systems? This problem has been studied in some depth, most notably in the literature on "minority games" [8] [9]. This line of work has investigated how to design agents so that their local decisions no longer interact to produce substantial resource use oscillations. One example involves designing agents that make resource selection decisions using historical resource utilization values [7]. If the agents look an appropriate distance into the past, they will be looking at the resource state one oscillation back in time, which should be a good approximation of the current resource utilization. The agent's delay parameter is tuned using survival of the fittest: agents with a range of delay factors are created, and the ones that get the highest utility survive and reproduce, while others do not. With this in place the resource utilization, under some conditions, settles down to near-optimal values.

Any approach predicated on the careful design of agent resource selection strategies, however, faces a fundamental flaw in an open systems context. In open systems, we do not control the design or operation of the consumer agents and can not be assured that they will adopt strategies that avoid emergent dysfunctions. Our challenge, therefore, is to find an approach that moderates or eliminates oscillatory resource utilization dynamics without needing to control the design or operation of the consumer agents.

Our Approach: Stochastic Request Rejection

Our approach to this problem is inspired by a scheme developed to improve the allocation of network router bandwidth (the resource) to client computers (the consumers). Routers currently operate as follows **<ref>**. Clients send packets to routers, and routers then forward the packets on towards their final destination. Every router has a buffer where it stores the packets that arrive when the router is busy. If any packets arrive when the buffer is full, the routers send a 'packet drop' message to the originating clients, at which point they immediately drop their data send rate to some minimal value, and then gradually increase it. This scheme is prone to inefficient router use oscillations because clients can synchronize their data send rate changes. If several get a packet dropped message from the router at about the same time, they all slow down and potentially under-utilize the router until their data rates ramp up enough to overload it, at which point they all drop their rates again.

A scheme called "Random Early Detect" (RED) has been proposed to address this problem [10]. The idea is simple. Rather than dropping packets only when the buffer is full, the router drops them stochastically with a probability proportional to how full the buffer is (e.g. 50% full results in a 50% chance of a packet being dropped). RED is successful at damping oscillations in router utilization because it de-synchronizes the data send rate changes across clients. While it does increase client-side message traffic by increasing the number of reject messages, it has no negative impact on total throughput because it is very unlikely to cause the buffers to empty out and leave a router needlessly unutilized.

We have adapted this idea for request-based resource sharing in open agent systems. We call our technique 'stochastic request rejection', or SRR. Imagine that every resource stochastically rejects new requests with a probability proportional to its current load. This can be implemented by the resource itself, or by 'sentinel' agents that track the number of consumers each resource is currently serving, and stochastically intercept/reject consumer requests with a probability proportional to that load. When oscillations occur, we would predict that the increased level of rejections from the currently more heavily utilized resource will shift the requests to the less-utilized resource, thereby damping the oscillations and ameliorating their negative impact on the utility achieved by consumer agents.

Experimental Evaluation: The Grocery Checkout Case

Our first set of tests studied the value of SRR when applied to "grocery checkout" resource sharing scenario. The experimental setup was as follows. There were 20 consumers and 2 resources. Each consumer sends a 'request' message to the resource it believes has the smallest backlog, waits until it receives a 'job completed' message from the resource, and then after a randomized delay sends the next 'request' message. The consumers' estimate of a resources' utility may lag the correct value. Resources may either take on requests or reject them. If a consumer receives a 'reject' message, it sends the request to the other resource. Messages take 20 units of time to travel, resources require 20 units of time to perform each task, and consumers have a normally distributed delay at 40 ticks, with a standard deviation of 10, between receiving one result and submitting the next request. The aggregate results reported below represent averages over 100 simulation runs, each 4000 ticks long, and all our conclusions were statistically significant at $p < 0.01$.

The impact of applying SRR in this scenario can be visualized as follows:

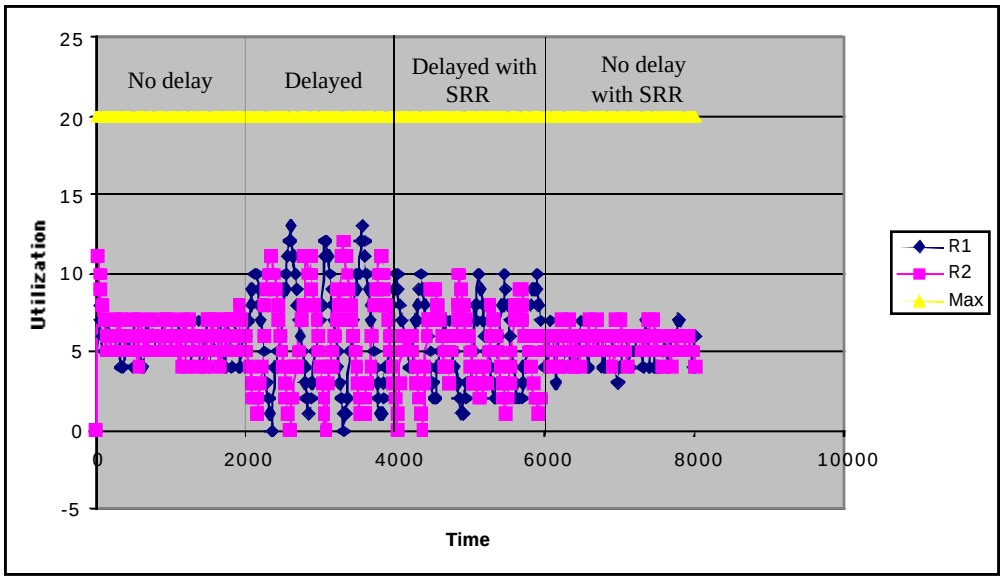


Figure 4. The impact of SRR on resource oscillations.

In this simulation run, the agents initially made their resource requests using current information on the length of each resources' backlog. As we can see, in this case the resource utilization clusters tightly around the optimal distribution of 50-50 across resources. At T = 2000, the backlog information provided to the consumers was made 100 time units out of date, rapidly leading to large resource use oscillations. At T = 4000, SRR was turned on, resulting in substantial damping in the magnitude of these oscillations. At T = 6000, the delay was removed but SRR was left on, whereupon the

resource utilization returns to clustering tightly around the optimal distribution. The aggregate results confirm the patterns suggested by this example:

	Null	**SRR**
No delay	160 +/- 4 0%	160 +/- 6 33%
Short Delay (50)	160 +/- 7 0%	160 +/- 6 34%
Long Delay (100)	167 +/- 8 0%	161 +/- 6 35%

Table 2. Task completion times +/ 1 standard deviation, as well as reject rates, for different delays, with and without SRR.

As we would expect for the grocery store scenario, the variability in task completion times without SRR increases with the delay in status information, and if the delay is long enough, the average task completion time can increase as well. If we turn on SRR, we find that it significantly reduces the variability in task completion times in the delayed cases, and almost eliminates the increase in task completion times in the long delay case. Rejecting some requests can thus actually speed up task completion when delay-induced oscillations occur. But this does come at a cost. Message traffic is increased: roughly 1/3rd of the consumer requests elicit a reject message and must be re-sent. The variability of task completion times in the no delay case is also increased by SRR. This is because many resource requests that would otherwise simply have been queued up incur the additional delay of being rejected and re-submitted. The absolute size of these effects can be expected to vary with the ratio of task and messaging times. Ideally, we would be able to enable SRR only when it is needed, so we can avoid incurring its costs in the no-oscillation contexts where it is not helpful. We will return to this point later.

Experimental Evaluation: The Movie Theatre Case

The parameters for the movie theatre simulations were the same as the grocery store case, except for the following changes. Resources do not have a waiting line, but instead offer concurrent access to 15 different ‘slots’ with varying utility (the first slot has value 15, the second has value 14, and so on). Tasks take 160 ticks to perform. The aggregate results for this case are as follows:

	Null	SRR
No delay	9.6 +/- 1.5 0% 331	9.7 +/- 1.2 59% 303
Short Delay (50)	9.1 +/- 1.9 0% 332	9.8 +/- 1.4 60% 303
Long Delay (100)	7.6 +/- 2.1 3% 331	9.6 +/- 1.4 66% 300

Table 3. Average quality +/- 1 standard deviation, as well as reject rates and number of completed requests, for different delays, with and without SRR.

As we can see, SRR is also effective in this scenario. Delay-induced oscillations cause consumers to often select the resource that is actually more heavily utilized and thus lower in quality, resulting in a reduction of the average achieved quality. Using SRR eliminates this problem, but with the cost of increasing message traffic, as well as reducing the rate of task completion (since every time a task is rejected a delay is incurred while the request is re-submitted). As in the "grocery checkout" case, we would ideally prefer to be able to apply SRR selectively, so we do not incur these costs when oscillations are not occurring.

Avoiding Needless Rejects Via Selective SRR

It is in fact straightforward to use spectral analysis to determine if persistent oscillations are occurring in resource utilization. In our implementation, each resource periodically (every 20 ticks) sampled its utilization and submitted the last 30 data points to a Fourier analysis. SRR was turned on if above-threshold values were encountered in the power spectrum so determined. The threshold was determined empirically. This approach proved to be successful. In the grocery checkout scenario, selective SRR was as effective as SRR in maintaining throughput and task duration consistency while avoiding increases in message traffic in the no-delay case:

	Null	SRR	Selective SRR
No delay	160 +/- 4 0%	160 +/- 6 33%	160 +/- 4 0%
Short Delay (50)	160 +/- 7 0%	160 +/- 6 34%	160 +/- 6 29%
Long Delay (100)	167 +/- 8 0%	161 +/- 6 35%	161 +/- 6 33%

Table 4. Task completion times +/ 1 standard deviation, as well as reject rates, for different delays, with and without [selective] SRR.

In the movie theatre scenario, selective SRR was as effective as SRR in maintaining task quality while almost eliminating increases in message traffic and task completion time in the no-delay case:

	Null	SRR	Selective SRR
No delay	9.6 +/- 1.5 0% 331	9.7 +/- 1.2 59% 303	9.5 +/- 1.4 6% 327
Short Delay (50)	9.1 +/- 1.9 0% 332	9.8 +/- 1.4 60% 303	9.6 +/- 1.5 41% 311
Long Delay (100)	7.6 +/- 2.1 3% 331	9.6 +/- 1.4 66% 300	9.3 +/- 1.6 54% 305

Table 5. Average quality +/- 1 standard deviation, as well as reject rates and number of completed requests, for different delays, with and without [selective] SRR.

This simple spectral analysis approach can be fooled, of course, into triggering SRR when resource use oscillations are due to variations in aggregate demand, as opposed to status information delays. But this problem is easily addressed: whenever a resource detects significant usage oscillations, it analyzes the correlation of it's utilization with that of the other resource. Variations in aggregate demand will show a positive correlation, while delay-caused oscillations show a negative one. We have implemented this approach and found that it successfully avoids triggering SRR for aggregate demand variations while remaining effective in responding to delay-induced oscillations.

Contributions and Next Steps

We have presented a promising approach for mitigating the deleterious effects of delay-induced resource-use oscillations on request-based resource sharing. It differs from previous techniques in that it is designed to be appropriate for the important domain of open systems, where we can not rely on being able to control the design or operation of the resource consumers. The key elements of this approach involve the stochastic load-proportional rejection of resource requests, triggered selectively when spectral and cross-resource correlation analyses reveal that delay-induced oscillations are actually taking place.

Next steps for this work include evaluating the selective SRR approach when there are more than two resources. This research is part of the author's long-standing efforts to develop a systematic enumeration of the different multi-agent system exception types as well as how they can be addressed in open systems contexts [11] [12]. See http://ccs.mit.edu/klein/ for further details on this work.

Acknowledgements

This work was supported by the NSF Computational and Social Systems program as well as the DARPA Control of Agent-Based Systems program.

References

1. Jensen, D. and V. Lesser. *Social pathologies of adaptive agents*. In the proceedings of *Safe Learning Agents Workshop in the 2002 AAAI Spring Symposium*. 2002: AAAI Press.
2. Chia, M.H., D.E. Neiman, and V.R. Lesser. *Poaching and distraction in asynchronous agent activities*. In the proceedings of *Proceedings of the Third International Conference on Multi-Agent Systems*. 1998. Paris, France.
3. Youssefmir, M. and B. Huberman. *Resource contention in multi-agent systems*. In the proceedings of *First International Conference on Multi-Agent Systems (ICMAS-95)*. 1995. San Francisco, CA, USA: AAAI Press.
4. Sterman, J.D., *Learning in and about complex systems*. 1994, Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts Institute of Technology. 51.
5. Klein, M., H. Sayama, P. Faratin, and Y. Bar-Yam, *The Dynamics of Collaborative Design: Insights From Complex Systems and Negotiation Research.* Concurrent Engineering Research & Applications, 2003. **In press**.
6. Waldrop, M., *Computers amplify Black Monday.* Science, 1987. **238**: p. 602-604.
7. Hogg, T., *Controlling chaos in distributed computational systems.* SMC'98 Conference Proceedings, 1998(98CH36218): p. 632-7.
8. Challet, D. and Y.-C. Zhang, *Emergence of Cooperation and Organization in an Evolutionary Game.* arXiv:adap-org/9708006, 1997. **2**(3).

9. Zhang, Y.-C., *Modeling Market Mechanism with Evolutionary Games.* arXiv:cond-mat/9803308, 1998. **1**(25).
10. Braden, B., D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, *Recommendations on Queue Management and Congestion Avoidance in the Internet*. 1998, Network Working Group.
11. Klein, M. and C. Dellarocas. *Exception Handling in Agent Systems*. In the proceedings of *Proceedings of the Third International Conference on AUTONOMOUS AGENTS (Agents '99)*. 1999. Seattle, Washington.
12. Klein, M., J.A. Rodriguez-Aguilar, and C. Dellarocas, *Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death.* Autonomous Agents and Multi-Agent Systems, 2003. **7**(1/2).

Appendix E: Using Domain-Independent Exception Handling Services to Enable Robust Open Multi-Agent Systems: The Case of Agent Death

MARK KLEIN, PHD
Center for Coordination Science
Massachusetts Institute of Technology
m_klein@mit.edu
http://ccs.mit.edu/klein/

JUAN ANTONIO RODRIGUEZ-AGUILAR, PHD
Center for Coordination Science
Massachusetts Institute of Technology
Jarjar@mit.edu
http://ccs.mit.edu/ases/jar/

CHRYSANTHOS DELLAROCAS, PHD
Sloan School of Management
Massachusetts Institute of Technology
dell@mit.edu
http://ccs.mit.edu/dell

Abstract

This paper addresses a critical question involved in the development of multi-agent systems: how can we create robust systems out of the often unreliable agents and infrastructures we can expect to find in open systems contexts? We propose an approach to this problem based on distinct exception handling (EH) services that enact coordination protocol-specific but domain-independent strategies to monitor agent systems for problems ('exceptions') and intervene when necessary to avoid or resolve them. The value of this approach is demonstrated for the 'agent death' exception in the well-known Contract Net protocol; we show through simulation experiments that the EH service approach provides substantially improved performance compared to existing approaches in a way that is appropriate for open multi-agent systems.

1. The Challenge: Enabling Robust Open Multi-Agent Systems

"Open systems ... represent arguably the most important application for multi-agent systems" [1]

This paper addresses one simple question: how can we develop robust multi-agent systems out of the often unreliable (buggy, malicious, or simply "dumb") agents and infrastructures we can expect to encounter in open system contexts? This is becoming an increasingly critical question because of emerging changes in the way human organizations work. Globalization, enabled by ubiquitous telecommunications, has increasingly required that organizations be assembled and re-configured within small time frames, often bringing together partners that have never worked together before. Examples of this include international coalition military forces, disaster recovery operations, open electronic marketplaces and virtual supply chains [2] [3] [4]. Multi-agent systems (MAS) represent one of the most promising approaches for supporting these kinds of applications, because of their ability to use multi-agent coordination protocols to dynamically self-organize themselves as their problems and constituent agents change [5] [1]. A critical open challenge remains, however. The vast majority of MAS work to date has considered well-behaved agents running on reliable infrastructures in relatively simple domains [6]. These have been almost exclusively *closed* systems, i.e. where the agents and their infrastructure are developed and enacted under centralized control. It is clear however that these assumptions do not hold for the *open* contexts described above, where agents can come from multiple sources and must operate on the infrastructures at hand [7]. For these contexts we can expect, in contrast, to find:

- *Unreliable Infrastructures.* In large distributed systems like the Internet, unpredictable host and communication problems can cause agents to slow down or die unexpectedly, messages to be delayed, garbled or lost, etc. These problems become worse as the applications increase in size, due to the increase in potential points of failure.

- *Non-compliant agents*. In open systems, agents are developed independently, come and go freely, and can not always be trusted to follow the rules properly due to bugs or even outright malice. This can be expected to be especially prevalent and important in contexts such as electronic commerce or military operations where there may be significant incentives for fraud or malice.
- *Emergent dysfunctions*. Emerging multi-agent system applications are likely to involve complex and dynamic interactions that can lead to emergent dysfunctions,

such as chaotic behavior, with the multi-agent coordination mechanisms that have proved most popular to date [8] [9] [10]. This is especially true since agent societies operate in a realm where relative communication and computational costs and capabilities can be radically different from those in human society, leading to behaviors with which we have little previous experience. It has been argued, for example, that the 1987 stock crash was due in part to the action of computer-based "program traders" that were able to execute trade decisions at unprecedented speed and volume, leading to unprecedented stock market volatility [11].

All of these departures from "ideal" multi-agent system behavior can be called *exceptions*, and the results of inadequate exception handling include the potential for poor performance, system shutdowns, and security vulnerabilities.

2. Our Approach: Distinct Domain-Independent Exception Handling Services

It is certainly imaginable that agents could be individually elaborated so that they could handle all exceptions they are apt to face, and most agent system exception handling research has in fact taken this direction. Even one of the first MAS coordination protocols, the Contract Net, included an "immediate response bid', which allowed an agent to determine whether the exception of getting no bids for its tasks was due to all eligible subcontractor agents being busy (in which case a retry is appropriate) or due to the outright lack of subcontractors with the necessary skills (in which case presumably the system manager/user should be informed) [12]. This "survivalist" approach to multi-agent exception handling faces, however, a number of serious shortcomings:

First of all, it greatly increases the burden on agent developers. It is predicated upon implementing potentially complicated and carefully coordinated exception handling behaviors in all agents. Developers must anticipate and correctly prepare for all the exceptions the agent may encounter, which is problematic at best since the agent's operating environments may be difficult to anticipate. Making changes in exception handling behavior is difficult because it potentially requires coordinated changes in multiple agents created by different developers. Agents become harder to maintain, understand and reuse because a potentially large body of exception handling code obscures the relatively simple normative behavior of an agent.

Perhaps more seriously, this approach can result in poor exception handling performance. In open systems it is always possible that some agents will not comply properly with these more sophisticated protocols or may violate some of their underlying assumptions. Some exception handling approaches, for example, are based on game-theoretic incentive

analyses [13] that assume all agents are fully rational and share a particular class of utility function (typically profit maximization), but this obviously may not always be the case. Some agents may be buggy, face severe computational limitations that preclude full rationality, or have radically different utility functions (e.g. cause as much damage to a particular vendor as possible). All agent interactions are potentially slowed down by the overhead incurred by the more heavyweight 'exception-savvy' protocols. Some kinds of interventions (such as "killing" a broken or malicious agent) may in addition be difficult to implement because the agents do not have the established legitimacy needed to apply such interventions to their peers. Finally, finding the appropriate responses to some kinds of exceptions (notably emergent exceptions) often requires that the agents achieve a more or less global view of the multi-agent system state, which is notoriously difficult to create without heavy bandwidth requirements.

It is in order to address these limitations that we have been defining an approach that enhances MAS robustness by offloading exception handling from problem solving agents to distinct, domain-independent services. We call this the "citizen" approach by analogy to the way exceptions are handled in human society. In such contexts, citizens typically adopt relatively simple and optimistic rules of behavior, and rely on a whole host of social institutions (the police, lawyers and law courts, disaster relief agencies, the Security and Exchange Commission, the Better Business Bureau, and so on) to handle most exceptions. This is generally a good tradeoff because such institutions are able, by virtue of specialized expertise, widely accepted legitimacy, and economies of scale, to deal with exceptions more effectively and efficiently than individual citizens, while making relatively few demands of most agents (e.g. pay your taxes, obey police officers, report crimes).

The key insight that makes the "citizen" approach workable in the multi-agent system context is the simple but powerful notion that highly reusable, *domain-independent* exception handling expertise can be usefully separated from the knowledge used by agents to do their "normal" work. There is substantial evidence for the validity of this claim. Early work on expert systems development revealed that it is useful to separate domain-specific problem solving and generic control knowledge [14, 15]. Analogous insights were also confirmed in the domains of collaborative design conflict management [16, 17] and workflow exception management [18]. In our work to date we have found that every coordination protocol has its own characteristic set of domain-independent exceptions, which in turn can be mapped to domain-independent strategies potentially applicable for handling (anticipating and avoiding, or detecting and resolving) them. We shall see some examples of such strategies below; for others please see [19] [20].

3. Case Study: Handling Agent Death in the Contract Net Protocol

Let us illustrate this approach by considering how it can be applied to an important exception scenario: handling agent death in the Contract Net protocol (henceforth called CNET) [12]. CNET is a market-based protocol for allocating tasks to agents. It is probably the most widely-used MAS coordination protocol and has been applied successfully to many domains including manufacturing control [21], tactical simulations [22], transportation scheduling [23], and distributed sensing [24].

The CNET protocol operates as follows (Figure 1):

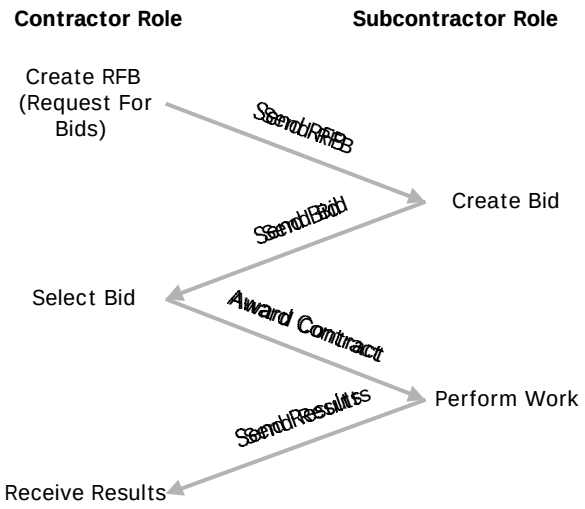


Figure 1. A simple version of the Contract Net protocol.

An agent (hereafter called the “contractor”) identifies a task that it cannot or chooses not to do locally and attempts to find another agent (hereafter called the “subcontractor”) to perform the task. It begins by creating a Request For Bids (RFB) which describes the desired work, and then sending it to potential subcontractors (typically identified using a matchmaker that indexes agents by the skills they claim to have). Interested subcontractors respond with bids (specifying such issues as the time needed to perform the task) from which the contractor selects a winner. The winning agent, once notified of the award, performs the work (potentially subcontracting out its own subtasks as needed) and submits the results to the contractor. CNET is prone to a wide range of potential exceptions from all three of the categories (unreliable infrastructures, non-compliant agents, emergent dysfunctions) described above [25].

Let us consider what happens when a CNET agent 'dies'. Agent death can be common in a large distributed system. Even the most carefully crafted code has been estimated to include an average of three bugs, mostly intermittent ones, per 1000 lines of code [26], and this does not account for the possibility that malicious agents may intentionally 'kill' themselves in order to sabotage the efforts of their customers. If a CNET agent dies there are several immediate consequences. If the agent is acting as a subcontractor, its customer obviously will not receive the results it is expecting. In addition, if the agent has subcontracted out one or more subtasks, these subtasks and all the sub-sub-… tasks created to achieve them become "orphaned", in the sense that there is no longer any purpose for them, so they are uselessly tying up potentially scarce resources. Finally, if the system uses a matchmaker, it will continue to offer the now dead agent as a candidate (a "false positive"), resulting in wasted message traffic and perhaps fostering the illusion that agents with particular skills are available in a multi-agent system when they no longer are. CNET agent death presents a surprisingly rich source of challenges and helps reveal, we believe, many of the important issues involved in exception handling in open agent systems.

The standard mechanism used to handle agent death in CNET, as in many distributed protocols, is a classic "survivalist" approach: timeout/retry. If no results are received by the deadline the subcontractor promised, a contractor will re-start the subcontracting process for that task, sending a new RFB. This approach does work but rather inefficiently, since it does not eliminate orphaned tasks, does not remove false positives from the matchmaker, and is prone to an "timeout cascade" effect, wherein the death of an agent performing a subtask can cause cascading timeouts and retries for its customers, the customers of its customers, and so on, all the way up to the CNET agent at the top of the task decomposition tree, resulting in needless delays and wasted work.

Contrast this with a "citizen"-style approach to handling the agent death exception. In our implementation of this approach, when an agent joins the MAS, the EH service begins periodic polling of the agent. If an agent dies (does not respond to polling in a timely way), the EH service takes a series of coordinated actions to resolve the problem

- It notifies the matchmaker that this agent is dead and should therefore be removed from the list of available subcontractors. This handles false matchmaker positives.
- If the dead agent was performing tasks for some customer(s), the EH service immediately asks these customers to re-allocate the tasks assigned to the dead agent. This avoids the "timeout cascade" effect described above, since contractors only reallocate tasks when the subcontractor has actually died.

- If the dead agent had allocated tasks to other agents, the EH service tries to find new customers for these orphaned tasks by acting in effect as a proxy. The proxy waits for an RFB for the orphaned tasks, and submits a bid that is likely to be highly competitive since the tasks are either already in process or actually completed. This is a reasonable strategy in domains where there is a standardized task decomposition, so the replacement for the dead agent is apt to require the same subtasks that the dead agent did. If the proxy wins the anticipated RFB, it forwards task results as they are generated. Otherwise it keeps responding to RFBs until it wins or the task results become obsolete. This strategy thus minimizes the work wasted on orphaned tasks. In domains where the proxy approach in inappropriate (e.g. results get obsolete very quickly, or there is no standard task decomposition) the EH service can simply kill all orphaned tasks.
- An agent reliability database is notified so it can keep up to date information about the mean time between failures for each agent type.

In addition to this, the EH service can help *avoid* agent death problems exception via bid filtering. Whenever a contractor sends out an RFB, the EH service can transparently filter out the bids that come from the most failure-prone of the bidders, thereby reducing the probability that a task will be assigned to an agent that dies during its enactment.

The EH service makes two assumptions about agents in order to provide these capabilities. One is that it can transparently monitor and, if necessary, modify the domain-independent aspects (message types as well as task and agent IDs) of all inter-agent messages. This is straightforward to achieve if the EH service is realized using "sentinels" integrated into the communications infrastructure (Figure 2).

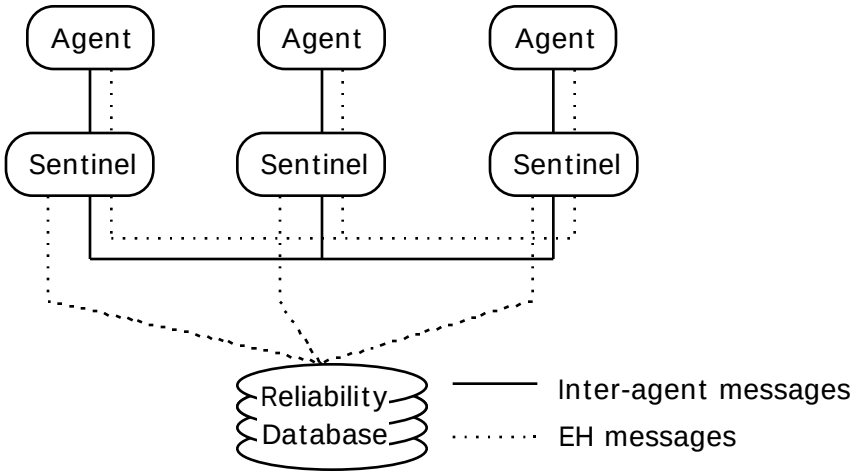


Figure 2: Sentinel architecture for EH service.

In this architecture, every agent (including the matchmaker if any) is "wrapped" with a sentinel through which all of its in- and out-going message traffic is routed. Sentinels can communicate with each other as well as with the agent reliability database. The distributed nature of this architecture allows us to avoid performance and reliability bottlenecks. The overhead of passing messages through sentinels can be minimized if sentinels are located on the same hosts as their agent "clients". Most EH messages go between a client and its sentinel; messages are interchanged between sentinels only when killing orphaned tasks. The reliability database is accessed only when an agent dies or a sentinel is filtering a set of bids, and standard distributed database techniques can be used to avoid bottlenecks and reliability problems in accessing/updating this information [26]. Sentinel death can be dealt with using techniques such as that described in [27].

The second assumption is that, when agents enter a multi-agent system supported by the EH service, they indicate the kinds of exception handling behavior they can support. This 'EH signature' specifies for that agent how agent death can be detected (i.e. whether or not that agent responds to the "are you alive?" message), how dead subcontractor problems are resolved (i.e. whether or not an agent responds to the "resend RFB" message), how dead customer problems are resolved (i.e. whether the agent allows orphaned task proxying and/or responds to the "cancel task" message), and how dead subcontractor problems are avoided (i.e. whether or not the agent allows bid filtering). 'Full' citizens support all options, while pure survivalists support none. Other agent types come somewhere in between. This allows the EH services to account for agent heterogeneity.

Note that we are not claiming that this particular architecture and set of agent death handling strategies is the optimal, or even the only way in which agent death can be offloaded to an EH service. Our claim, rather, is that, at least for this particular exception, the citizen approach can provide significant advantages over survivalist approaches to exception handling in open multi-agent systems.

4. Evaluating the Exception Handling Services Approach

We ran a series of experiments to test this claim in a multi-agent system running the CNET protocol. The experiments all take place in a discrete event based MAS simulator built on top of the Swarm Simulation System [28]. The scenario consists of several dozen CNET agents, one per host, interacting over a reliable network. Contractor agents send out an RFB with a specified timeout period: potential subcontractors bid only if they become available during this period (i.e. subcontractors perform only one task at a time).

Bids are binding, which means that subcontractors will bid on a new RFB only after the timeout for its pending bid expired without an award being received (presumably because some other subcontractor won the task). Contractors select the winning bids based solely on how quickly the bidders claimed they could perform the task. Contractors re-send RFBs if no bids have been received by the timeout period (presumably because no subcontractors with the needed skills were available at that time). This CNET protocol is modeled on the one described in [12] and was chosen because it is simple and was shown to represent a reasonable design tradeoff in several test domains.

We designed the experiments to evaluate exception handling performance in a range of domain types. Three independent variables were selected to capture what we judged to be key domain-dependent elements affecting exception handling performance: task tree topology, task length, and agent scarcity. Task completion performance was measured for four different agent configurations, whose parameters are summarized below:

Configuration	**# of Agents**	**Task duration**[3]
Short tasks, abundant subcontractors	50	10
Short tasks, scarce subcontractors	16	100
Long tasks, abundant subcontractors	50	10
Long tasks, scarce subcontractors	16	100

In all of these configurations there were three top-level contractors, each executing a loop wherein they announce a new top-level task, wait for bids, award the contract to the best bidder, wait to receive the results and then repeat the above steps. Every top-level task involved the completion of task trees with depth 4 and branching factor 2, thereby requiring the combined contribution of 15 agents (Figure 3)

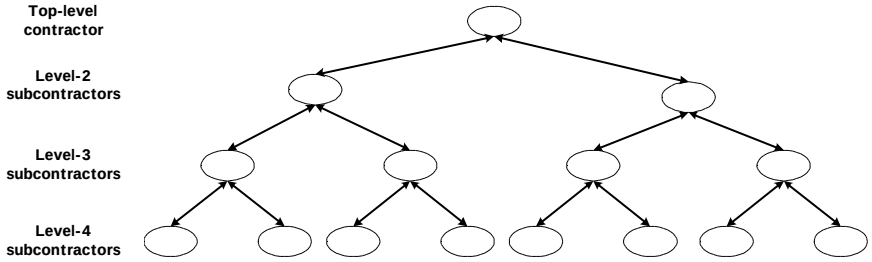


Figure 3. Top-level tasks require the creation of a 4-level task tree.

[3] The task duration is the number of cycles that a contractor has to spend *after* it has received the results of its subcontracted tasks, before it returns *its* result to *its* contractor. The task duration given is relative to the RFB timeout: short tasks are only 10 times longer than the RFB timeout, while long tasks are 100 times longer.

This allowed us to study the effects of the EH service for tree topologies of differing depths, ranging from "flat" (one level of decomposition, as in a client-server setup) to "deep" (three levels of decomposition, as we might expect to find in more complex information supply chains). Tree width was not varied because it does not affect total task completion time for any particular agent death instance. To simplify the experiment, all subcontractors were capable of performing any task in a given task tree.

In our initial set of experiments, three simulation runs were performed for each of the four configurations described above:

- Failure-free environment (baseline case)
- Failure-prone environment, "survivalist" agents using timeout/retry.
- Failure-prone environment, "citizen" agents fully supported by the EH services

In the failure-prone cases, subcontractor agents were divided into three reliability classes. All subcontractor agents had a "lifespan" (time until death) selected randomly from a geometric distribution with mean time between failures (MTBF) equal to 10 times the task duration for low reliability agents, 50 times the task duration for medium reliability agents, and 100 times the task duration for high reliability agents. When an agent dies, a new one with the same skills and reliability class but a different unique ID is created and registered with the matchmaker. This is done to keep the subcontractor population from shrinking over the course of the experiment, thereby emulating a large and dynamic agent pool where the population of subcontractors remains roughly constant. All simulations were run until a 90% confidence interval could be computed for each of the completion time estimates with a width of less than 15 percent of the estimated mean.

Figure 4 below summarizes the mean task completion times for deep task trees, normalized relative to the failure-free (baseline) case, for survivalist and citizen agents in each of the four agent configurations described above.

Figure 4. Normalized mean task completion times.

As expected, citizen agents supported by the EH services produced task completion times in agent death affected trees greatly superior to those of survivalist agents. In the deep tree case shown above, mean completion times for citizen agents were as much as 3.4 times faster than for survivalist agents. Remarkably, citizen agents gave times for failure-affected trees no more than 27% longer than the failure-free mean. The advantage of citizen agents was much less dramatic for shallow task trees (not shown above): they were only about 50 to 60% faster than survivalist agents. This is because the 'timeout cascade' effect that plagues survivalist but not citizen agents only appears for deeper task decompositions. The size of the citizen advantage was only mildly affected by task length and subcontractor scarcity for the configurations tested.

We also noted that citizen agents had, for most conditions, a much smaller variation in task completion times than did survivalist agents (Figure 5):

Figure 5. Normalized standard deviations of task completion times.

The variance reduction represents, we believe, a significant benefit since in many environments we can expect that consistency will be equally as important as efficiency. A system with a low mean completion time, but where some task instances may take a very long time to complete is bound to make some users extremely unhappy.

The benefits of the citizen approach are achieved in a way that is well-suited to open multi-agent systems. Recall that the most salient aspect of open systems is that we can make only minimal assumptions about the agents that compose it, since they were not developed under centralized control. Agents are thus likely to be heterogeneous with respect to their exception handling behavior. The EH service can work with a broad range of such behaviors by using the 'EH signature' concept described above. It requires at most that agents support three very simple directives ("are you alive?", "resend RFB", and "cancel-task") and can provide significant support for agents (e.g. survivalists) that implement none of these messages, since bid filtering and orphaned task proxying operate in a way that is completely transparent to the affected agents. The EH service can use timeouts to detect agent death if the agent does not support the "are you alive?" message, avoiding timeout cascades by being aware of the inter-agent task commitment structure. The EH service can notify an agents' customer to resend an RFB if the agent

itself does not support the “resend-RFB” message. Finally, if an agent does not support “cancel-task”, at worse we are unable to avoid wasting some computational resources. We can expect that the degree of benefits derived from the EH service will be a function of which subset of the full ‘EH signature’ the agents support, which is born out by our experiments (Figure 6):

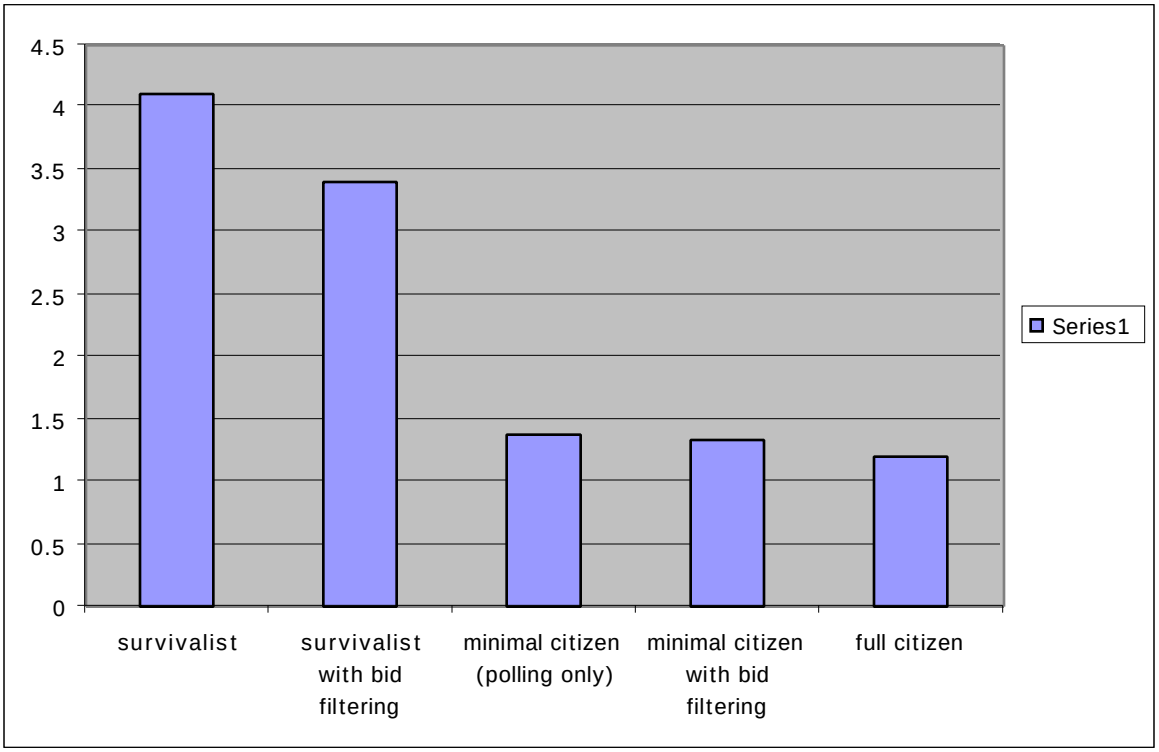


Figure 6. Normalized mean task completion times as a function of EH options supported.

These experiments explored the mean task completion time performance for the “long tasks, abundant subcontractors, deep trees” configuration for agents that support differing subsets of the maximal EH contract, normalized relative to the failure free case. As we can see, mean completion times decrease as the scope of the EH signature supported by the agent increases, and the ‘full citizen’ (which supports polling, proxying and bid filtering) is the fastest.

5. Contributions of this Work

The contributions of this work can be viewed at two levels. Narrowly construed, this work presents a powerful new approach for improving failure tolerance in open systems contexts. There is of course a substantial body of previous work on failure tolerance in

distributed systems, which has produced such useful techniques such as mirroring and rollbacks [29] [30], but these techniques achieve generality at the cost of efficiency and are not fully suited to open system contexts. Mirroring techniques are only appropriate for agents that were designed explicitly to keep a backup process in a synchronized state, but this clearly may not be true for all agents in open systems. Rollback techniques assume that all components cooperate with rollback directives if any failures are detected, and tend to be highly inefficient for deep task decompositions because full rollbacks can discard a lot of already completed work in such contexts. Our approach in effect prescribes a set of techniques, including carefully delimited partial rollbacks in addition to proxying and bid filtering, specifically designed to operate efficiently for open systems of agents that use CNET (or similar market-based mechanisms) for task allocation. Kumar et al. [27] present an approach that focuses on replacing dead middle agents with their interchangeable peers. This does not address, however, many of the problems (e.g. orphaned tasks) or potential solutions (e.g. proxying or bid filtering) that arise in task allocation processes, and is a closed system approach as it makes the strong assumption that all middle agents participate fully in a joint intentions framework.

More broadly speaking, this work presents a detailed example of the potential value of a domain-independent EH services approach to increasing robustness in open agent systems. This represents, we believe, a significant contribution to previous efforts in this area. As we have already noted, there has been relatively little previous work on multi-agent exception handling, and much of it has taken a "survivalist" approach with the important shortcomings identified above. Several lines of research have begun to explore concepts similar to those presented here, but none as far as we know have explored the combination of domain-independent exception handling implemented as distinct services. Hägg [6] presents the concept of sentinel agents; these are distinct services, which monitor the agent system and intervene when necessary by selecting alternative problem solving methods, excluding faulty agents, or reporting to human operators. This approach is not domain-independent, however: sentinels must be customized for each new application. Kaminka et al [31] present Social Attentive Monitoring (SAM), an exception handling approach wherein agents detect exceptions via uncovering violations of normative relationships with their teammates, and exploit a teamwork model to diagnose and fix these problems. This approach does have generic elements, but it is limited to teamwork protocols like TEAMCORE [32] and requires domain-dependent customization of the exception detection procedures. Horling et al. [33] have explored the use of domain-independent tools to detect and resolve the exception wherein the agents have a harmfully inaccurate picture of the inter-agent dependencies in their current context. This approach is limited to a single exception type, however, and like SAM applies to just one class of coordination protocol. Venkatraman et al [34] describe a

generic approach to uncovering agents that do not comply with coordination protocols. This approach only addresses one subclass of exception types, however, and does not include a resolution component. Related work can also be found if we go farther afield into such disciplines as planning, distributed systems, manufacturing process control, and the like. There has also been substantial work in the planning and robotics communities on dealing with unexpected world states [35] [36] [37] [38] [39] [40]. This work focuses almost exclusively on exceptions (e.g. failed operations, unexpected events) in the world manipulated by the agents, and not on exceptions concerning the agents themselves. Finally, there has been substantial work on detecting and resolving exceptions in computer-supported cooperative work [41] [42] [43] [44] [45] and manufacturing control [46] [47] [48] but this has been applied to a very limited range of domains (e.g. just flexible manufacturing cell control) and exception types (e.g. just inappropriate task assignments).

6. Future Work

We plan to pursue two concurrent lines of development in this work. One line will include defining and evaluating individual techniques for handling other important open MAS exceptions within the EH service framework. We are currently focusing on exceptions characteristic of market-based resource allocation protocols, including reputation fraud (i.e. an exception that occurs due to agents trying to 'fool' a reputation server), contract violations, and emergent dynamical dysfunctions (i.e. thrashing).

A second line of work will be to increase the power and scope of the domain-independent EH services that apply these individual techniques in the context of a particular MAS. In a MAS it will often be necessary to diagnose a particular presenting problem (e.g. a late task) in order to determine the underlying cause (e.g. agent death, overloaded agents, or missing agent skills) and thereby identify the appropriate exception handler to use. Model-based diagnosis is potentially applicable, but must face novel challenges because, by definition, agents in open systems are black boxes for which we may not have the detailed behavioral models assumed by current model-based diagnosis techniques.

The long-term goal is to integrate these efforts to create domain-independent knowledge-based exception handling services with the following functional architecture (Figure 7):

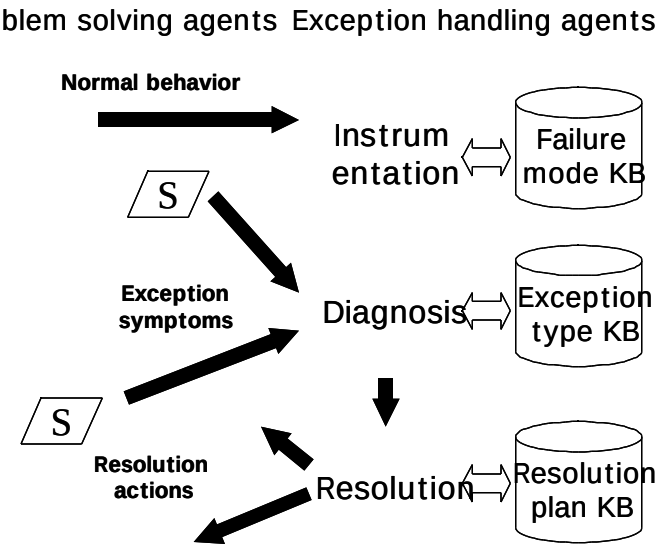


Figure 7. Functional architecture for the envisioned exception handling service.

In this vision, when new agents enter a MAS they register with the EH service, establishing a 'social contract' that specifies their normative behavior, rights and responsibilities within the system. The EH service then consults its failure mode knowledge base to determine the appropriate techniques needed to avoid or detect the characteristic exceptions this agent may face or create. When an exception does occur, the EH service diagnosis the underlying cause, and then selects and enacts the techniques suitable for resolving this particular problem.

Acknowledgements

This work was supported by NSF grant IIS-9803251 (Computation and Social Systems Program) and by DARPA grant F30602-98-2-0099 (Control of Agent Based Systems Program).

References

[1] M. Wooldridge, N. R. Jennings, and D. Kinny, "A Methodology for Agent-Oriented Analysis and Design," presented at Proceedings of the Third Annual Conference on Autonomous Agents (AA-99), Seattle WA USA, 1999.

[2] "Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces,". Edinburgh, Scotland, 1999.

[3] K. Fischer, J. P. Muller, I. Heimig, and A.-W. Scheer, "Intelligent agents in virtual enterprises.," presented at Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96), Blackpool, UK, 1996.

[4] M. B. Tsvetovatyy, M. Gini, B. Mobasher, and Z. Wieckowski, "MAGMA: An agent-based virtual marketplace for electronic commerce," *Applied Artificial Intelligence*, vol. 11, pp. 501-524, 1997.

[5] N. R. Jennings, K. Sycara, and M. Wooldrige, "A Roadmap of Agent Research and Development," *Autonomus Agents and Multi-Agent Systems*, vol. 1, pp. 275-306, 1998.

[6] S. Hägg, "A Sentinel Approach to Fault Handling in Multi-Agent Systems," presented at Proceedings of the Second Australian Workshop on Distributed AI, in conjunction with Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96), Cairns, Australia, 1996.

[7] C. Hewitt and P. D. Jong, "Open Systems," Massachusetts Institute of Technology, MIT AI Lab Technical Report 1982.

[8] M. Youssefmir and B. Huberman, "Resource contention in multi-agent systems," presented at First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, CA, USA, 1995.

[9] J. D. Sterman, *Learning in and about complex systems*. Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1994.

[10] M. H. Chia, D. E. Neiman, and V. R. Lesser, "Poaching and distraction in asynchronous agent activities," presented at Proceedings of the Third International Conference on Multi-Agent Systems, Paris, France, 1998.

[11] M. Waldrop, "Computers amplify Black Monday," *Science*, vol. 238, pp. 602-604, 1987.

[12] R. G. Smith and R. Davis, "Distributed Problem Solving: The Contract Net Approach," *Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence*, 1978.

[13] T. Sandholm, S. Sikka, and S. Norden, "Algorithms for Optimizing Leveled Commitment Contracts," presented at Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999.

[14] T. R. Gruber, "A Method For Acquiring Strategic Knowledge," *Knowledge Acquisition*, vol. 1, pp. 255-277, 1989.

[15] J. A. Barnett, "How Much Is Control Knowledge Worth? A Primitive Example," *Artificial Intelligence*, vol. 22, pp. 77-89, 1984.

[16] M. Klein, "Conflict Resolution in Cooperative Design," in *PhD thesis. Computer Science.* Urbana-Champaign, IL.: University of Illinois, 1989.

[17] M. Klein, "Supporting Conflict Resolution in Cooperative Design Systems," *IEEE Systems Man and Cybernetics*, vol. 21, pp. 1379-1390, 1991.

[18] M. Klein, "Exception Handling in Process Enactment Systems," MIT Center for Coordination Science, Cambridge MA, CCS Working Paper 203, December 1997 1997.

[19] M. Klein and C. Dellarocas, "Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems," Massachusetts Institute of Technology, Cambridge MA USA, Center for Coordination Science Working Paper CCS-WP-211, http://ccs.mit.edu/papers/pdf/wp211.pdf, May 2000 2000.

[20] M. Klein and C. Dellarocas, "Towards a Systematic Repository of Knowledge about Managing Multi-Agent System Exceptions," Massachusetts Institute of Technology, Cambridge MA USA, ASES Working Paper ASES-WP-2000-01, http://ccs.mit.edu/klein/papers/ASES-WP-2000-01.pdf, February 2000 2000.

[21] A. Baker, "Complete manufacturing control using a contract net: a simulation study," presented at Proceedings of the International Conference on Computer Integrated Manufacturing, Troy New York USA, 1988.

[22] K. Boettcher, D. Perschbacher, and C. Wessel, "Coordination of distributed agents in tactical situations," *Ieee*, pp. 1421-6, 1987.

[23] M. Bouzid and A.-I. Mouaddib, "Cooperative uncertain temporal reasoning for distributed transportation scheduling," *Proceedings International Conference on Multi Agent Systems*, 1998.

[24] R. G. Smith and R. Davis, "Applications Of The Contract Net Framework: Distributed Sensing," *Distributed Sensor Nets: Proceedings of a Workshop*, 1978.

[25] C. Dellarocas and M. Klein, "An Experimental Evaluation of Domain-Independent Fault Handling Services in Open Multi-Agent Systems," presented at Proceedings of The International Conference on Multi-Agent Systems (ICMAS-2000), Boston, MA, 2000.

[26] J. Gray and A. Reuter, *Transaction Processing : Concepts and Techniques*. San Mateo, Calif. USA: Morgan Kaufmann Publishers, 1993.

[27] S. Kumar, P. R. Cohen, and H. J. Levesque, "The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams," presented at Proceedings of the International Conference on Multi-Agent Systems (ICMAS-2000), Boston MA USA, 2000.

[28] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, "The Swarm Simulation System: A Toolkit for Building Multi-Agent Systems," Santa Fe Institute, Santa Fe, New Mexico, USA Working Paper 96-06-042, 1996.

[29] A. Burns and A. Wellings, *Real-Time Systems and Their Programming Languages*: Addison-Wesley, 1996.

[30] S. J. Mullender, *Distributed systems*, 2nd ed. New York; Wokingham, England ; Reading, Mass.: ACM Press. Addison-Wesley Pub. Co., 1993.

[31] G. A. Kaminka and M. Tambe, "What is Wrong With Us? Improving Robustness Through Social Diagnosis," presented at Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), 1998.

[32] M. Tambe, "Towards flexible teamwork," *Journal of Artificial Intelligence Research*, vol. 7, pp. 83-124, 1997.

[33] B. Horling, V. Lesser, R. Vincent, A. B. A, and P. Xuan, "Diagnosis as an Integral Part of Multi-Agent Adaptability," University of Massachussets at Amherst Department of Computer Science, Amherst, Massachussets, Technical Report 99-03, 1999.

[34] M. Venkatraman and M. P. Singh, "Verifying Compliance with Commitment Protocols: Enabling Open Web-Based Multiagent Systems," *Autonomous Agents and Multi-Agent Systems*, vol. 3, 1999.

[35] P. Traverso, L. Spalazzi, and F. Giunchiglia, "Reasoning about acting, sensing and failure handling: a logic for agents embedded in the real world," *Intelligent Agents II. Agent Theories, Architectures, and Languages. IJCAI`95 Workshop*, 1996.

[36] A. E. Howe, "Improving the reliability of artificial intelligence planning systems by analyzing their failure recovery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, pp. 14-25, 1995.

[37] L. Birnbaum, G. Collins, M. Freed, and B. Krulwich, "Model-Based Diagnosis of Planning Failures," presented at Proceedings of the National Conference on Artificial Intelligence (AAAI-90), 1990.

[38] C. A. Broverman and W. B. Croft, "Reasoning About Exceptions During Plan Execution Monitoring," presented at Proceedings of the National Conference on Artificial Intelligence (AAAI-87), 1987.

[39] R. J. Firby, "An Investigation into Reactive Planning in Complex Domains," presented at Proceedings of AAAI-87, 1987.

[40] K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Meyer, "Failure, monitoring and recovery in the agent language 3APL," *Cognitive Robotics. Papers from the*, 1998.

[41] P. Mi and W. Scacchi, "Articulation: An Integrated Approach to the Diagnosis, Replanning and Rescheduling of Software Process Failures," presented at Proceedings of 8th Knowledge-Based Software Engineering Conference, Chicago, IL, USA, 1993.

[42] D. K. W. Chiu, K. Karlapalem, and Q. Li, "Exception Handling in ADOME Workflow System," Hong Kong University of Science and Technology, Hong Kong, technical report Technical Report, 1997.

[43] M. Klein, "A Knowledge-Based Approach to Handling Exceptions in Workflow Systems," MIT Center for Coordination Science, Cambridge MA USA, CCS Working Paper 203, April 1998 1998.

[44] E. Auramaki and M. Leppanen, "Exceptions and office information systems," presented at Proceedings of the IFIP WG 8.4 Working Conference on Office Information Systems: The Design Process., Linz, Austria, 1989.

[45] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency Handling in Multi-perspective Systems," *IEEE Transactions on Software Engineering*, vol. 20, pp. 569-578, 1994.

[46] M. Fletcher and D. S. Misbah, "Task rescheduling in multi-agent manufacturing," *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA*, vol. 99, pp. 689-94, 1999.

[47] E. Adamides and D. Bonvin, "Failure recovery of flexible production systems through cooperation of distributed agents," *Ifip Transactions B: Computer Applications in Technology*, vol. 11, pp. 227-38, 1993.

[48] D. M. Katz, S., "Exception management on a shop floor using online simulation," presented at Proceedings of 1993 Winter Simulation Conference - (WSC '93), Los Angeles, CA, USA, 1993.

Appendix F: Protocols for Negotiating Complex Contracts

Mark Klein	Peyman Faratin	Hiroki Sayama	Yaneer Bar-Yam
Massachusetts Institute of Technology	Massachusetts Institute of Technology	University of Electro-Communications	New England Complex Systems Institute
m_klein@mit.edu	peyman@mit.edu	sayama@hc.uec.ac.jp	yaneer@necsi.org

Abstract

Work to date on negotiation protocols has focused on defining contracts consisting of one or a few independent issues. Many real-world contracts, by contrast, are much more complex, consisting of multiple inter-dependent issues and intractably large contract spaces. This paper describes a simulated annealing based approach appropriate for negotiating such complex contracts that achieves near-optimal outcomes for negotiations with binary issue dependencies.

Keywords: non-linear mediated single text unmediated proposal exchange negotiation, multiple interdependent issues, prisoner's dilemma

1. Introduction

Work to date on negotiation protocols has focused on negotiating what we can call 'simple' contracts, i.e. contracts consisting of one or a few independent issues. These protocols work via the iterative exchange of proposals and counter-proposals. An agent starts with contract that is optimal for that agent and makes concessions, in each subsequent proposal, until either an agreement is reached or the negotiation is abandoned because the utility of the latest proposal has fallen below the agents' reservation value (Figure 1):

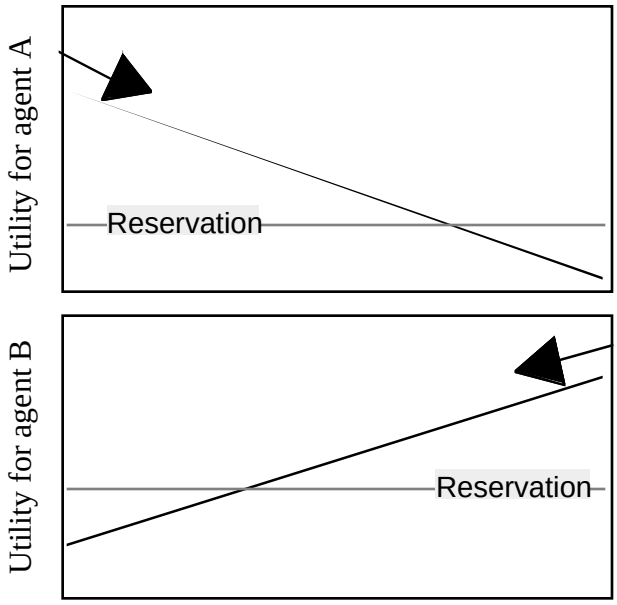


Figure 1: The proposal exchange model of negotiation, applied to a simple contract. The Y axis represents the utility of a contract to each agent. Each point on the X axis represents a possible contract, ordered in terms of its utility to agent B. Since there is no need to negotiate over issues that both parties agree upon, we only consider issues where improvement for one party represents a decrement for the other. The arrows represent how agents begin with locally optimal proposals, and concede towards each other, with their subsequent proposals, as slowly as possible. Note that we have, for presentation purposes, 'flattened' the contract space onto a single dimension, but there should actually be one dimension for every issue in the contract.

This is a perfectly reasonable approach for simple contracts. Since issues are independent, the utility of a contract for each agent can be calculated as the weighted sum of the utility for each issue. The utility function for each agent is thus a simple one, with a single optimum and a monotonic drop-off in utility as the contract diverges from that ideal. Simple contract negotiations thus typically progress as follows:

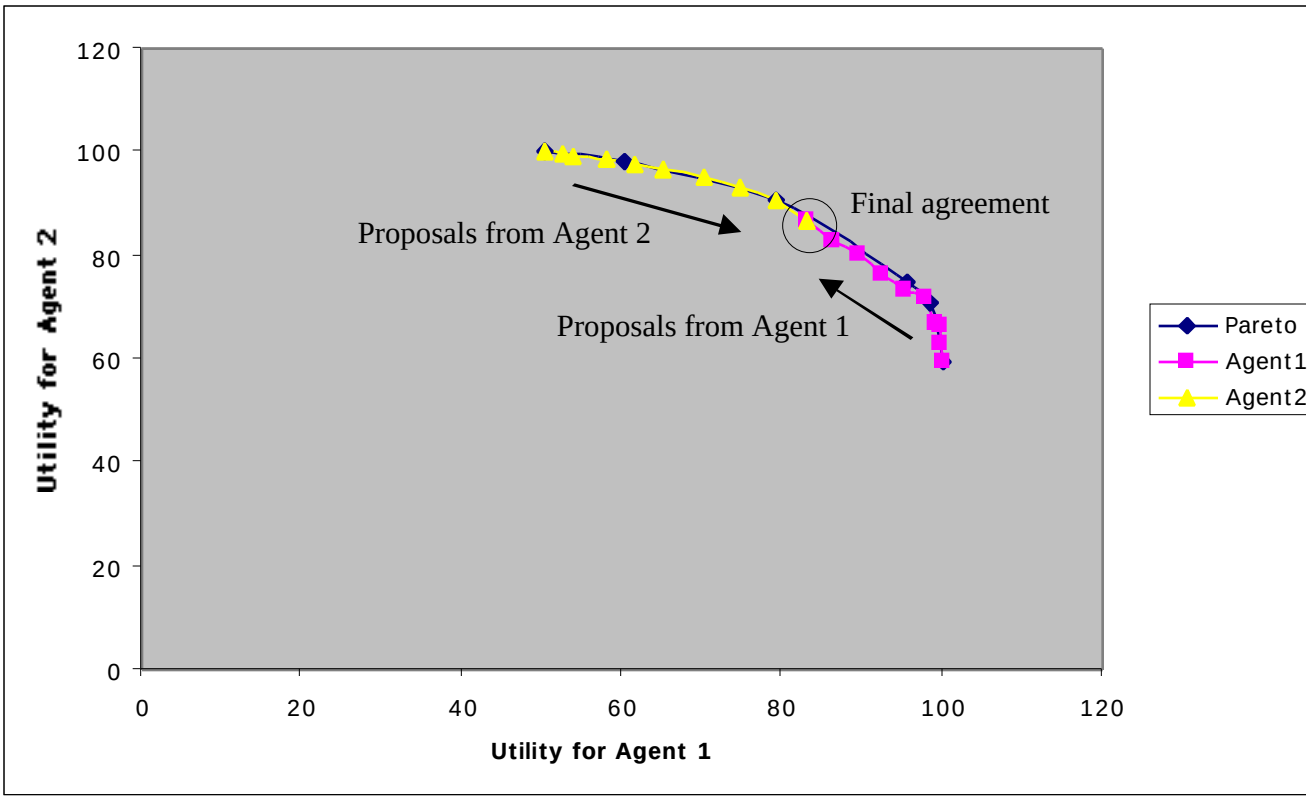


Figure 2. The utilities for the proposals made in a typical simple contract negotiation. The contract consisted in this case of 40 binary issues. Each agent starts with a locally optimal proposed contract (at the extremes of the Pareto frontier) and is required to reduce the Hamming distance (number of issues with different values) between the two agents' proposals, until an agreement is reached. With simple contracts, this results in optimal outcomes. The Pareto frontier, representing the set of optimal contracts, was estimated by applying an annealing optimizer to differently weighted sums of the two agents' utility functions.

As we can see, the proposals from each agent start at their own ideal, and then track the Pareto frontier until they meet in the middle with an optimal agreement. This happens because, with linear utility functions, it is easy for an agent to identify the proposal that represents the minimal concession: the contract that is minimally worse than the current one is "next" to the current one in the contract space and can be found by moving in the direction with the smallest aggregate utility slope. The simplicity of the utility functions, moreover, makes it feasible for agents to infer enough about their opponents that they can identify concessions that are attractive to each other, resulting in relatively quick negotiations.

Real-world contracts, by contrast, are generally much more complex, consisting of a large number of inter-dependent issues. A typical contract may have tens or even hundreds of distinct issues. Even with only 50 issues and two alternatives per issue, we encounter a search space of roughly 10^15 possible contracts, too large to be explored exhaustively. The value of one issue selection to an agent, moreover, will often depend on the selection made for another issue. The value to me of a given couch, for example, depends on whether it is a good match with the chair I plan to purchase with it. Such issue interdependencies lead to nonlinear utility functions with *multiple* local optima [1]:

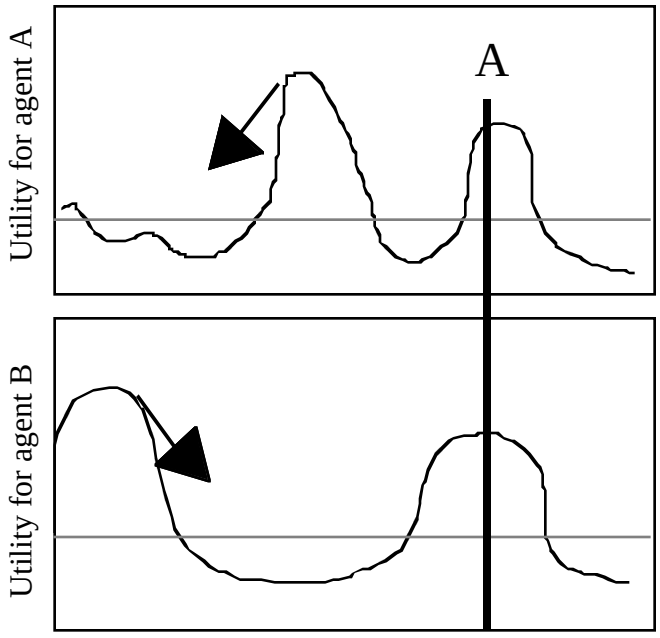


Figure 3: An example of proposal exchange applied to a complex contract. Because of issue inter-dependencies, the utility functions have multiple optima. The arrows show what happens when each agent begins at a local optimum and concedes towards the other: win-win solutions (such as that represented by contract A) found elsewhere in the contract space can be missed.

In such contexts, an agent finding its own ideal contract becomes a nonlinear optimization problem, difficult in its own right. Simply conceding toward the other agents' proposals can result in the agents missing contracts that would be superior from both their perspectives (e.g. the contract labeled "A" in figure 3 above). Standard negotiation techniques thus typically produce the following behavior when applied to complex contract negotiation:

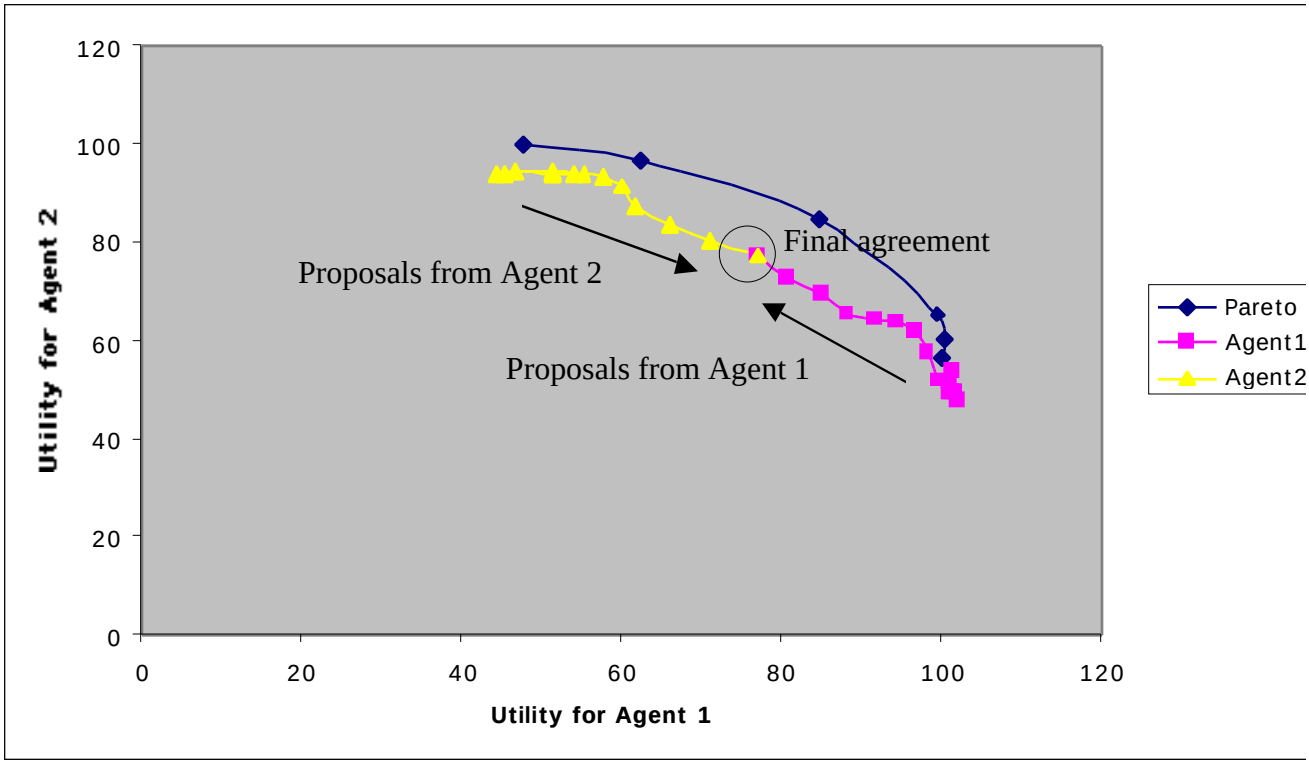


Figure 4. The utilities for the proposals made in a typical complex contract negotiation. This example differs from figure 2 only in that a nonlinear utility function was used by each agent (details below). As we can see, the minimal concession protocol that works optimally for simple contracts produces outcomes, for complex contracts, that are substantially sub-optimal.

The agents start with an approximation to their ideal contract and diverge increasingly from the Pareto frontier as they converge upon an agreement. The degree of sub-optimality depends on the details of the utility function. In our experiments, for example, the final contracts' averaged 94% of optimal. This is a substantial decrement when you consider that the utility functions we used for each agent were, individually, quite easy to optimize: a simple steepest ascent search averaged final utility values roughly 97% of those reached by a nonlinear optimization algorithm. It is striking that such relatively forgiving multi-optima utility functions lead to substantially sub-optimal negotiation outcomes.

These sub-optimal outcomes represent a fundamental weakness with current negotiation techniques. The only way to ensure that subsequent proposals track the Pareto frontier, and thus conclude with a Pareto optimal result, is to be able to identify the proposal that represents the minimal concession from the current one. But in a utility function with

multiple optima, that proposal may be quite distant from the current one, and the only way to find it is to exhaustively enumerate all possible contracts. This is computationally infeasible, however, due to the sheer size of the contract space. Since the utility functions are quite complex, it is in addition no longer practical for one agent to infer the other's utility function. Complex contracts therefore require different negotiation techniques which allow agents to find 'win-win' contracts in intractable multi-optima search spaces in a reasonable amount of time. In the following sections we describe a family of negotiation protocols that make substantial progress towards achieving these goals. The paper is structured as follows. We begin by describing how a well-known non-linear optimization technique (simulated annealing) can be integrated with the mediated single text negotiation protocol to produce an approach that offers near-optimal outcomes for complex contract negotiations. We reveal the prisoner's dilemma that results from this approach, and propose a refined protocol, based on parity-maintaining annealing mediator, that resolves that problem. We conclude with describing an unmediated version of the negotiation protocol that is also effective at producing near-optimal outcomes with complex contracts.

2. Mediated Single Text Negotiation

A standard approach for dealing with complex negotiations in human settings is the mediated single text negotiation [2]. In this process, a mediator proposes a contract that is then critiqued by the parties in the negotiation. A new, hopefully better proposal is then generated by the mediator based on these responses. This process continues, generating successively better contracts, until some agreed-upon stopping point (e.g. the reservation utility value is met or exceeded for both parties). We can visualize this process as follows:

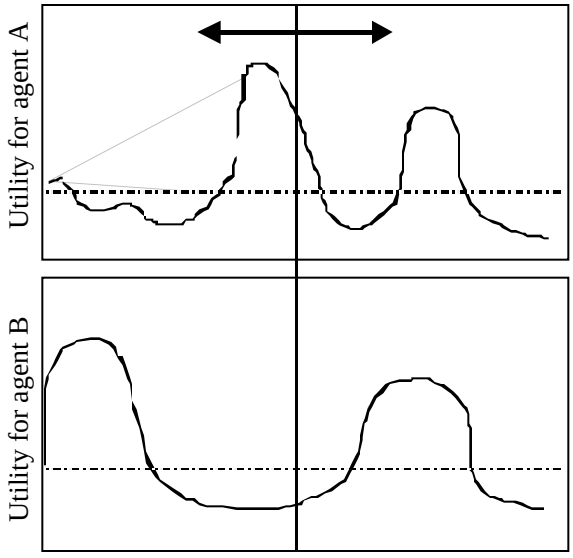


Figure 5: Single text negotiation. The vertical line represents the current proposed contract, and subsequent proposals move that line in the contract space.

Here, the vertical line represents the contract currently proposed by the mediator. Each new contract moves the line to a different point on the X axis. The goal is to find a contract that is sufficiently good for both parties.

We defined a simple simulation experiment to help us explore how well this approach actually works. In this experiment, there were two agents negotiating to find a mutually acceptable contract consisting of a vector S of 100 boolean-valued issues, each issue assigned the value 0 or 1, corresponding to the presence or absence of a given contract clause. This defined a space of 2^100, or roughly 10^30, possible contracts. Each agent had a utility function calculated using its own 100x100 influences matrix H, wherein each cell represents the utility increment or decrement caused by the presence of a given pair of issues, and the total utility of a contract is the sum of the cell values for every issue pair present in the contract:

$$U = \sum_{i=1}^{100} \sum_{j=1}^{100} H_{ij} S_j S_j$$

The influence matrix therefore captures the bilateral dependencies between issues, in addition to the value of any individual contract clause. For our experiments, the utility matrix was initialized to have random values between –1 and +1 in each cell. A different

influences matrix was used for each simulation run, in order to ensure our results were not idiosyncratic to a particular configuration of issue inter-dependencies.

The mediator proposes a contract that is initially generated randomly. Each agent then votes to accept or reject the contract. If both vote to accept, the mediator mutates the contract (by randomly flipping one of the issue values) and the process is repeated. If one or both agents vote to reject, a mutation of the most recent mutually accepted contract is proposed instead. The process is continued for a fixed number of proposals. Note that this approach can straightforwardly be extended to a N-party (i.e. multi-lateral) negotiation, since we can have any number of parties voting on the contracts.

We defined two kinds of agents: 'hill-climbers' and 'annealers'. The hill-climbers use a very simple decision function: they accept a mutated contract only if its utility to them is greater than that of the last contract both agents accepted. Annealers are more complicated. Each annealer has a virtual 'temperature' T, such that it will accept contracts worse than last accepted one with the probability:

$$P(\text{accept}) = \min(1, e^{-\Delta U/T})$$

where ΔU is the utility change between the contracts. In other words, the higher the virtual temperature, and the smaller the utility decrement, the greater the probability that the inferior contract will be accepted. The virtual temperature of an annealer gradually declines over time so eventually it becomes indistinguishable from a hill-climber. Annealing has proven effective in single-agent optimization, because it can travel through utility valleys on the way to higher optima [1]. This suggests that annealers can be more successful than hill-climbers in finding good negotiation outcomes.

3. The Prisoner's Dilemma

Negotiations with annealing agents did indeed result in substantially superior final contract utilities, but as the payoff table below shows, there is a catch:

	Agent 2 hill-climbs	Agent 2 anneals
Agent 1 hill-climbs	.86 .73/.74	.86 .99/.51
Agent1 anneals	.86 .51/.99	.98 .84/.84

Table 1: The optimality of the negotiation outcomes for different pairings of annealing and hill-climbing agents. The top value in each cell represents how close the social welfare value of the final contract is to optimal. The pair of values below it represent how close the final contract is to the optimum for the Agent 1 and Agent 2, respectively.

As expected, paired hill-climbers do relatively poorly while paired annealers do very well. If both agents are hill-climbers they both get a poor payoff, since it is difficult to find many contracts that represent an improvement for both parties. A typical negotiation with two hill-climbers looks like the following:

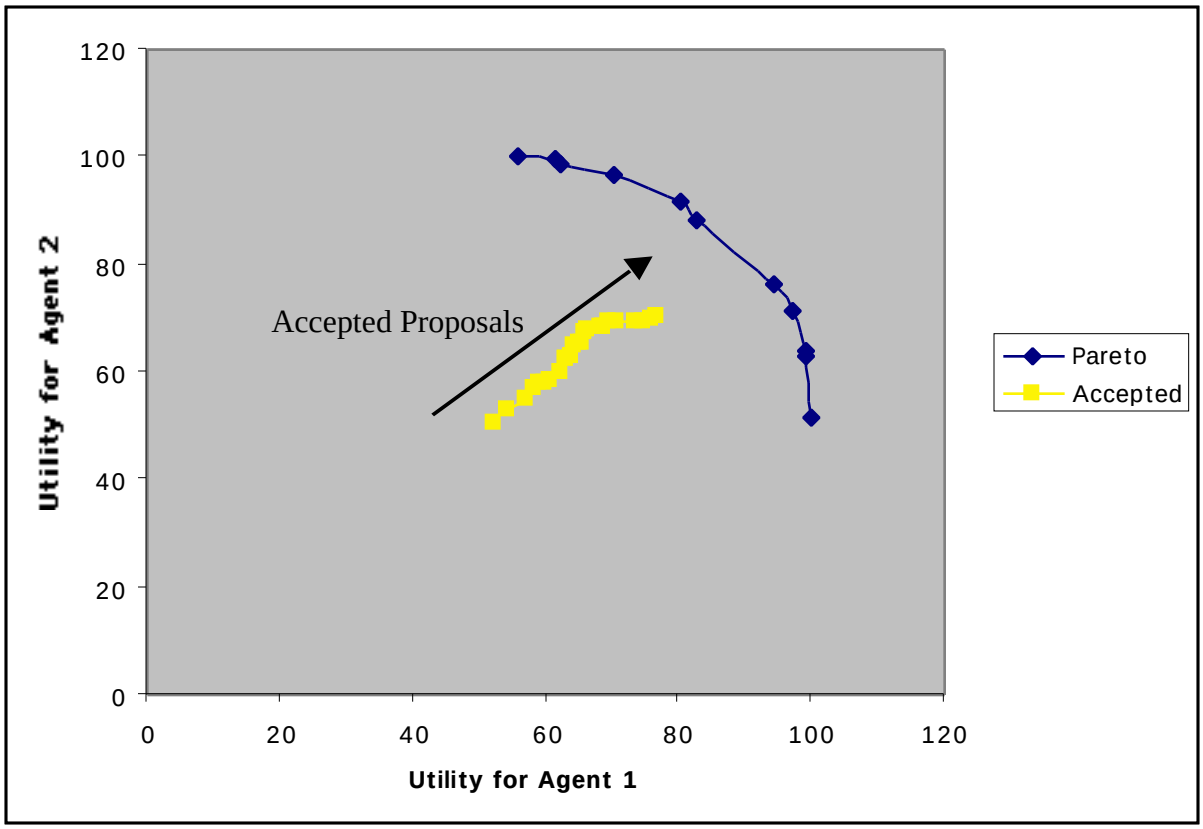


Figure 6: The utilities for the accepted proposals in a typical single text complex contract negotiation with two hill-climbers. The mediator's initial proposal is at the lower left, and the subsequent accepted proposals move towards higher utilities for both agents.

As we can see, in this case the mediator was able to find only a handful of contracts that increased the utility for both hill-climbers, and ended up with a poor final social welfare.

Near-optimal social welfare can be achieved, by contrast, when both agents are annealers, willing to initially accept individually worse contracts so they can find win-win contracts later on:

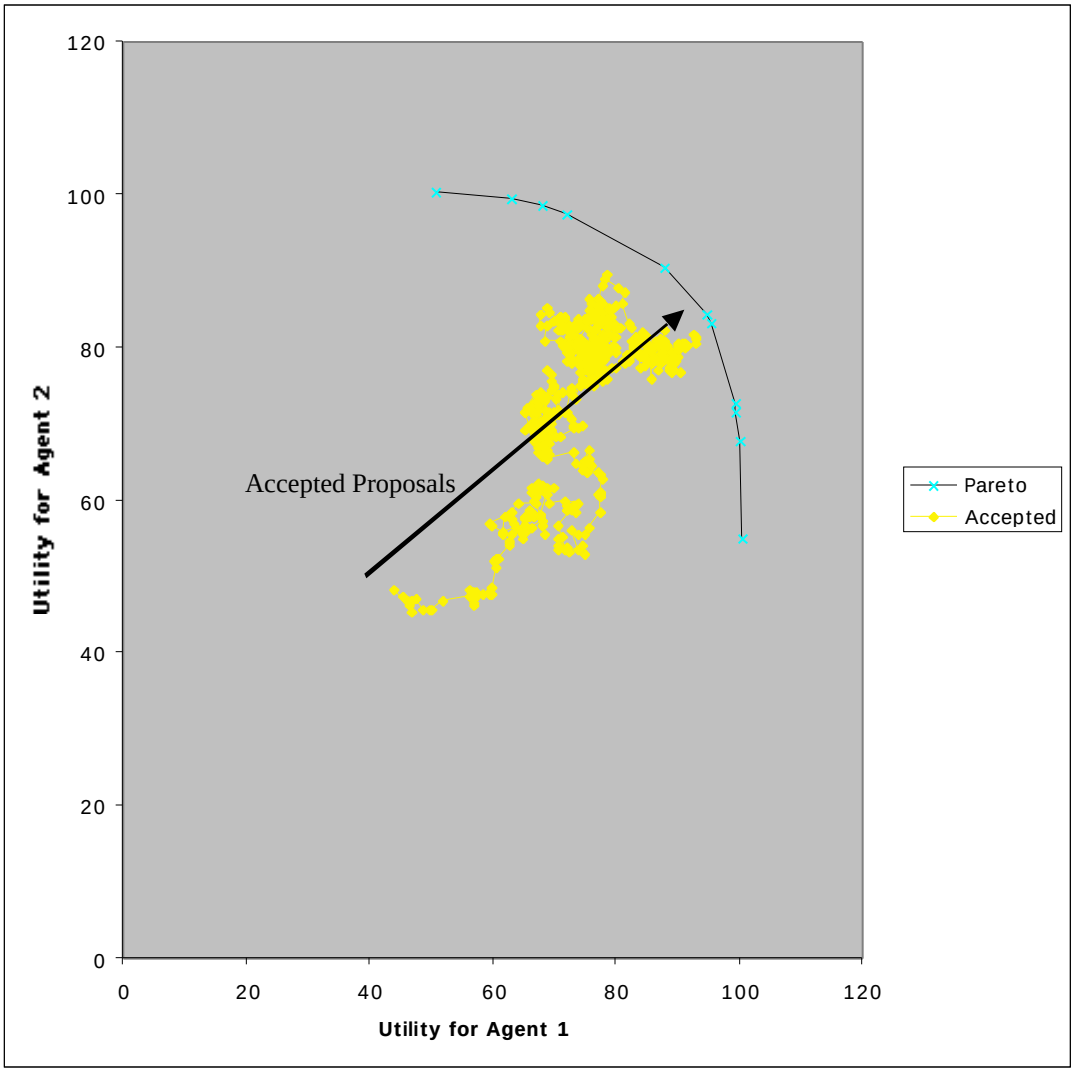


Figure 7: The utilities for the accepted proposals for a typical single text complex contract negotiation with two annealers. Some of the accepted proposals actually cause utility decrements for one or both agents, but the final result is a near-optimal contract.

The agents entertain a much wider range of contracts, eventually ending very near the Pareto frontier.

If one agent is a hill-climber and the other is an annealer, however, the hill-climber does extremely well but the annealer fares correspondingly poorly (Figure 8). This pattern can

be understood as follows. When an annealer is at a high virtual temperature, it becomes a chronic conceder, accepting almost anything beneficial or not, and thereby pays a "conceder's penalty". The hill-climber 'drags' the annealer towards its own local optimum, which is not very likely to also be optimal for the annealer:

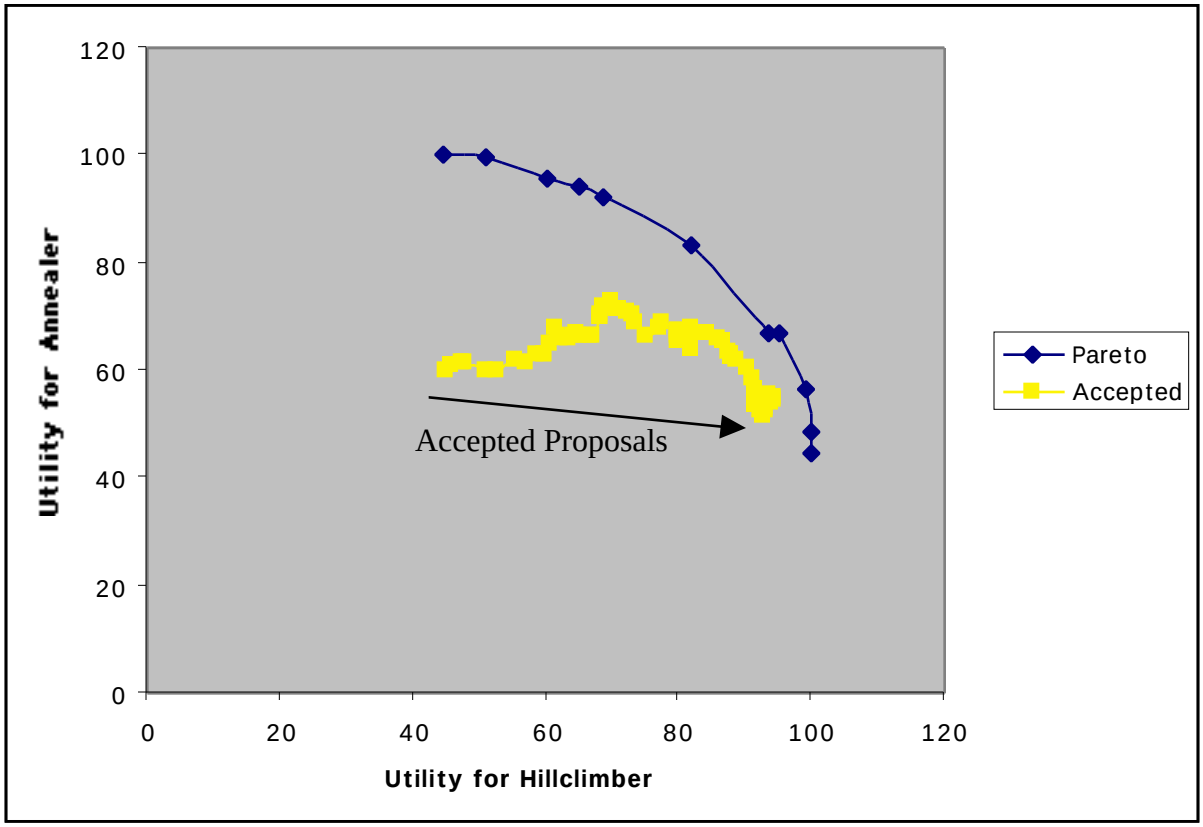


Figure 8: The utilities for the accepted proposals for a typical single text complex contract negotiation with an annealer and a hill climber. Note that the hill climber achieves a near-optimal contract at the expense of the annealer.

This reveals a dilemma. In negotiation contexts we typically can not assume that agents will be altruistic, and we must as a result design protocols such that the individually most beneficial negotiation strategies also produce the greatest social welfare [3]. In our case, however, even though annealing is a *socially* dominant strategy (i.e. annealing increases social welfare), annealing is not an *individually* dominant strategy. Hill-climbing is dominant, because no matter what strategy the other agent uses, it is better to be a hill-climber (Table I). If all agents do this, however, then they forego the higher individual utilities they would get if they both annealed. Individual rationality thus drive the agents towards the strategy pairing with the *lowest individual and social welfare*. This is thus an instance of the prisoner's dilemma. It has been shown that this dilemma can be avoided if

we assume repeated interactions between agents [4], but we would prefer to have a negotiation protocol that entices socially beneficial behavior without that difficult-to-enforce constraint. Several straightforward approaches to this problem, however, prove unsuccessful. One possibility is to simply reduce the annealer's willingness to make concessions. This can indeed eliminate the conceder's penalty, but at the cost of achieving social welfare values only slightly better than that achieved by two hill climbers. Another option is to have agents switch from being an annealer to a hill-climber if they determine, by observing the proposal acceptance rates of their opponents, that the other agent is being a hill-climber. We found, however, that it takes too long to determine the type of the other agent: by the time it has become clear, much of the contract utility has been committed, and it is too late to recover from the consequences of having started out as an annealer. See [5] for details.

4. The Annealing Mediator

We were able to define a negotiation protocol that avoids the prisoner's dilemma entirely in mediated single-text negotiation of complex contracts. The trick is simple: rather than requiring that the negotiating agents anneal, and thereby expose themselves to the risk of being dragged into bad contracts, we moved the annealing into the mediator itself. In our original protocol, the mediator would simply propose modifications of the last contract both negotiating agents accepted. In our refined protocol, the mediator is endowed with a time-decreasing willingness to follow up on contracts that one or both agents rejected (following the same inverse exponential regime as the annealing agents). Agents are free to remain hill-climbers and thus avoid the potential of making harmful concessions. The mediator, by virtue of being willing to provisionally pursue utility-decreasing contracts, can traverse valleys in the agents' utility functions and thereby lead the agents to win-win solutions. We describe the details of our protocol, and our evaluations thereof, below.

In our initial implementations each agent gave a simple accept/reject vote for each proposal from the mediator, but we found that this resulted in final social welfare values significantly lower than what we earlier achieved using annealing agents. In our next round of experiments we accordingly modified the agents so that they provide additional information to the mediator in the form of vote strengths: each agent annotates an accept or reject vote as being *strong* or *weak*. The agents were designed so that there are roughly an equal number of weak and strong votes of each type. This maximizes the informational content of the vote strength annotations. When the mediator receives these votes, it maps them into numeric values (strong accept = 1, weak accept = 0, weak reject = -1, strong reject = -2) and adds them together to produce an aggregate score. A proposal is accepted by the mediator if the score is non-negative, i.e. if both agents voted to accept it, or if a weak reject by one agent is overridden by a strong accept from the

other. The mediator can also accept rejected contracts (i.e. those with a negative aggregate score) using the annealing scheme described above. This approach works surprisingly well, achieving final social welfare values that average roughly 99% of optimal despite the fact that the agents each supply the mediator with only two bits of information. We found, in fact, that increasing the number of possible vote weights did not increase final social welfare. This is because the strong/weak vote annotations are sufficient to allow the system to pursue social welfare-increasing contracts that cause a utility decrement for one agent.

5. Incentives for Truthful Voting

Any voting scheme introduces the potential for strategic non-truthful voting by the agents, and our scheme is no exception. Imagine that one of the agents always votes truthfully, while the other exaggerates so that its votes are always 'strong'. One might expect that this would bias negotiation outcomes to favor the exaggerator and this is in fact the case:

	Agent 2 exaggerates	Agent 2 tells truth
Agent 1 exaggerates	.92 .81/.81	.93 .93/.66
Agent 1 tells truth	.93 .66/.93	.99 .84/.84

Table 2: The optimality of the negotiation outcomes for truth-telling vs exaggerating agents with a simple annealing mediator. An exaggeration strategy is individually enticed, even though it results in outcomes with lower social welfare.

As we can see, even though exaggerating has substantial negative impact on social welfare, agents are individually enticed to exaggerate, thus re-creating the prisoner's dilemma we encountered earlier. The underlying problem is simple: exaggerating agents are able to induce the mediator to accept all the proposals that are advantageous to them (if they are weakly rejected by the other agent), while preventing the other agent from doing the same. What we need, therefore, is an enhancement to the negotiation protocol that entices truthful voting, preserving equity and maximizing social welfare.

How can this be done? We found that simply placing a limit on the number of strong votes each agent can use does not work. If the limit is too low, we effectively lose the benefit of vote weight information and get the lower social welfare values that result. If the strong vote limit is high enough to avoid this, then all an exaggerator has to do is save

all of it's strong votes till the end of the negotiation, at which point it can drag the mediator towards making a series of proposals that are inequitably favorable to it.

Another possibility is to enforce overall parity in the number of "overrides" each agent gets. A override occurs when a contract supported by one agent (the "winner") is accepted by the mediator over the objections of the other agent. Overrides are what drags a negotiation towards contracts favorable to the winner, so it makes sense to make the total number of overrides equal for each agent. But this is not enough, because exaggerators always win disproportionately more than the truth-teller.

The solution, we found, came from enforcing parity between the number of overrides given to each agent *throughout* the negotiation, so neither agent can get more than a given advantage. This way at least rough equity is maintained no matter when (or whether) either agent chooses to exaggerate. The results of this approach were as follows when the override disparity was limited to 3:

	Agent 2 exaggerates	Agent 2 tells truth
Agent 1 exaggerates	.91 .79/.79	.92 .78/.81
Agent 1 tells truth	.92 .81/.78	.98 .84/.84

Table 3: The optimality of the negotiation outcomes for truth-telling vs exaggerating agents with parity-enforcing mediator. The parity-enforcing mediator makes truth-telling the rational strategy.

When we have truthful agents, we find that this approach achieves social welfare just slightly below that achieved by a simple annealing mediator, while offering a significantly ($p < 0.01$) higher payoff for truth-tellers than exaggerators. We found, moreover, that the same pattern of results holds for a range of exaggeration strategies, including exaggerating all the time, exaggerating at random, or exaggerating just near the end of the negotiation. Truth-telling is thus both the individually dominant and socially most beneficial strategy.

Why does this work? Why, in particular, does a truth-teller fare better than an exaggerator with this kind of mediator? One can think of this procedure as giving agents 'tokens' that they can use to 'purchase' advantageous overrides, with the constraint that both agents spend tokens at a roughly equal rate. Recall that in this case a truthful agent,

offering a mix of strong and weak votes, is paired with an exaggerator for whom at least some weak accepts and rejects are presented as strong ones. The truthful agent can therefore only get an override via annealing (see Table 3), and this is much more likely when its vote was a strong accept rather than a weak one. In other words, the truthful agent spends its tokens almost exclusively on contracts that truly offer it a strong utility increase. The exaggerator, on the other hand, will spend tokens to elicit a override even when the utility increment it derives is relatively small. At the end of the day, the truthful agent has spent its tokens more wisely and to better effect.

6. The Unmediated Single Text Protocol

The protocol we have just considered worked well in the contexts studied but suffers from the disadvantage of requiring a mediator. One issue concerns trust. Since the annealing mediator is empowered to selectively ignore agent votes, there is the risk that it may do so in a way that favors one agent over another (though the use of the parity-enforcing token mechanism does somewhat reduce the potential impact of this problem). Another issue concerns how quickly negotiations converge on a result. The annealing mediator generates new proposals by making random mutations to the last provisionally accepted contract, without taking into account any information about what contracts are preferable or even sensible. As a result, the mediator generates a very high proportion of rejected contracts, which is part of the reason why our experimental runs each involved so many (2500) proposals. The negotiating agents could imaginably provide the mediator with information about their utility functions so that the mediator is able to propose contracts more 'intelligently', but this is problematic for a number of reasons including the typical reluctance of self-interested agents to reveal their utility functions to a party that may or may not be worthy of their trust.

An effective unmediated version of the annealing protocol can, fortunately, be defined. It works as follows. Agents each start with a given number of tokens (2 each, in our experiments) and a mutually agreed-upon starting temperature T. A random contract is generated, and one of the negotiating agents is selected at random to propose a small (e.g. single-issue) variant thereof, presumably the variant that most increases the utility of the contract for that agent. The other agent then votes on the proposed variant. The proposing and voting both indicate the strength of their preference for the proposed contract using the scheme described above (i.e. strong reject, weak reject, weak accept, strong accept). The contract is provisionally accepted with probability

$$P(\text{accept}) = \min(1, e^{-\Delta U/T})$$

where the aggregate score (ΔU) is calculated as for the annealing mediator, and the outcome is determined using the roll of a fair, mutually observable dice. If the decision to accept a proposal represents the over-ride of one agents' reject vote, the winning agent needs to give one of its' tokens to the over-ridden agent. An over-ride is not permitted if the agent has run out of tokens. The proposer and voter alternate roles thereafter until neither agent can identify any improvements to make to the last accepted contract. Agents in the proposer role may pass but may not repeat proposals. The temperature T declines at a mutually agreed-upon rate during this process. This protocol thus reproduces the key elements of the annealing mediator protocol – a time-dependent annealing regime plus tokens - without the need for a mediator. Our experiments show that this protocol produces results just as good as the annealing mediator, averaging 99% of optimal, while requiring fewer proposal exchanges (averaging about 200 exchanges per negotiation).

7. Contributions

We have shown that negotiation involving complex contracts (i.e. those with many multiple inter-dependent issues) has properties that are substantially different from the simple (independent issue) case that has been studied to date in the negotiation literature, and requires as a result different protocols in order to achieve near-optimal outcomes. This paper presents, as far as we are aware, the first negotiation protocol designed specifically for complex contracts. While some previous work has studied multi-issue negotiation (e.g. [6] [7] [8]) the issue utilities in these efforts are treated as independent, so the utility functions for each agent are linear, with single optima. As we have seen, however, the introduction of multiple optima changes the game drastically. Multi-attribute auctions [9] represent another scheme for dealing with multiple issues, wherein one party (the buyer) publishes its utility function, and the other parties (the sellers) make bids that attempt to maximize the utility received by the buyer. If none of the bids are satisfactory, the buyer modifies its published utility function and tries again. This introduces a search process, and the problem with this approach is that it does not provide any guidance for how the parties involved should control their search through the vast space of possibilities. The essence of our own approach can be summarized simply: conceding early and often (as opposed to little and late, as is typical for independent issue negotiations) is a key to negotiating good complex contracts. Conceding is not individually rational in the face of agents that may choose not to concede, but this problem can be resolved either by introducing a mediator that stochastically ignores agent preferences, or by introducing dice into the negotiation protocol. In both cases, the exchange of tokens when one agent overrides another can be used to entice the truthful voting that enables win-win outcomes.

8. Next Steps

There are many other promising avenues for future work in this area. The high social welfare achieved by our approach partially reflect the fact that the utility functions for each agent, based as they are solely on binary dependencies, are relatively easy to optimize. Higher-order dependencies, common in many real-world contexts, are known to generate more challenging utility landscapes [10]. We hypothesize that it may be necessary to adapt non-linear optimization techniques such as genetic algorithms into the negotiation context in order to address this challenge. Another possibility involves agents providing limited information about their utility functions to the mediator or to each other in order to facilitate more intelligent search through very large contract spaces. Agents can, for example, tell the mediator which issues are heavily dependent upon each other, allowing the mediator to focus its attention within tightly-coupled issue 'clumps', leaving other less influential issues till later. We hypothesize that agents may be enticed to tell the truth in order to ensure that negotiations can complete in an acceptable amount of time. Finally, we would like to derive formal incentive compatibility proofs (i.e. concerning when agents are enticed to vote truthfully) for our protocols. New proof techniques will probably be necessary because previous results in this area have made strong assumptions concerning the shape of the agent utility functions that do not hold with complex contracts.

9. Acknowledgements

This work was supported by funding from the DARPA Control of Agent-Based Systems (CoABS) program, and the NSF Computation and Social Systems program.

10. References

1. Bar-Yam, Y., *Dynamics of complex systems*. 1997, Reading, Mass.: Addison-Wesley. xvi, 848.
2. Raiffa, H., *The art and science of negotiation.* 1982, Cambridge, Mass.: Belknap Press of Harvard University Press. x, 373.
3. Rosenschein, J.S. and G. Zlotkin, *Rules of encounter : designing conventions for automated negotiation among computers*. Artificial intelligence. 1994, Cambridge, Mass.: MIT Press. xxi, 229.
4. Axelrod, R., *The Evolution Of Cooperation*. 1984: Basic Books.
5. Klein, M., P. Faratin, and Y. Bar-Yam, *Using an Annealing Mediator to Solve the Prisoners' Dilemma in the Negotiation of Complex Contracts*, in *Proeedings of the Agent-Mediated Electronic Commerce (AMEC-IV) Workshop*. 2002, Springer-Verlag.
6. Faratin, P., C. Sierra, and N.R. Jennings, *Using similarity criteria to make negotiation trade-offs.* Proceedings Fourth International Conference on MultiAgent Systems. IEEE Comput. Soc., 2000.

7. Jonker, C.M. and J. Treur. *An Agent Architecture for Multi-Attribute Negotiation*. In the proceedings of *Proceedings of IJCAI-01*. 2001.
8. Fatima, S.S., M. Wooldridge, and N. R.Jennings. *MultiIssue Negotiation Under Time Constraints*. In the proceedings of *Proceedings of the 2002 International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*. 2002. Bologna, Italy: ACM.
9. Bichler, M. and J. Kalagnanam, *Bidding Languages and Winner Determination in Multi-Attribute Auctions.* European Journal of Operational Research, In Press.
10. Kauffman, S.A., *The origins of order: self-organization and selection in evolution*. 1993: Oxford University Press.

Appendix G: Exception Handling in Agent Systems

Mark Klein
Center for Coordination Science
MIT E40-169
Cambridge MA 02139
1 (617) 253-6796
m_klein@mit.edu

Chrysanthos Dellarocas
Sloan School of Management
MIT E53-315
Cambridge MA 02139
1 (617) 258-8115
dell@mit.edu

ABSTRACT

A critical challenge to creating effective agent-based systems is allowing them to operate effectively when the operating environment is complex, dynamic, and error-prone. In this paper we will review the limitations of current "agent-local" approaches to exception handling in agent systems, and propose an alternative approach based on a shared exception handling service that is "plugged", with little or no customization, into existing agent systems. This service can be viewed as a kind of "coordination doctor"; it knows about the different ways multi-agent systems can get "sick", actively looks system-wide for symptoms of such "illnesses", and prescribes specific interventions instantiated for this particular context from a body of general treatment procedures. Agents need only implement their normative behavior plus a minimal set of interfaces. We claim that this approach offers simplified agent development as well as more effective and easier to modify exception handling behavior.

Keywords

Exception failure handling detection and resolution

1. The Challenge: Exception-Capable Agent Systems

A critical challenge to creating effective agent-based systems is allowing them to operate effectively when, as is typical for many domains ranging from manufacturing to office work to military information gathering, the operating environment is complex, dynamic, and error-prone [1-5]. In such domains, we can expect to utilize a highly diverse set of agents; some have fairly sophisticated coordination capabilities, but many will be simple encapsulations of legacy applications. New tasks, agents and other resources can be expected to appear and disappear in unpredictable ways. Communication channels can fail or be compromised, agents can "die" (break down) or make mistakes, inadequate responses to the appearance of new tasks or resources can lead to missed opportunities or inappropriate resource allocations, unanticipated agent inter-dependencies can lead to

systemic problems like multi-agent conflicts, “circular wait” deadlocks, and so on. All of these departures from “ideal” collaborative behavior can be called *exceptions*. The result of inadequate exception handling is the potential for systemic problems such as clogged networks, wasted resources, poor performance, system shutdowns, and security vulnerabilities.

In this paper we will review the limitations of current “agent-local” approaches to exception handling in agent systems, and propose an alternative “shared service” approach that offers simplified agent development as well as more effective and easier to modify exception handling behavior. Initial versions of this service have been developed and tested in the multi-agent collaborative design conflict management domain; we will describe our preliminary results as well as our future plans.

2. Contributions and Limitations of Current Work

Current approaches to agent exception handling have serious limitations in terms of agent development cost and the effectiveness of system-wide exception handling behavior. The standard approach has been to “compile in” complicated and carefully coordinated exception handling behaviors into all problem-solving agents. This is, however, fundamentally problematic, since the causes, manifestations and resolutions for agent system exceptions are inherently systemic and context-sensitive rather than localizable to any particular agent. A circular wait deadlock, for example (where several agents are all stalled waiting for inputs from each other) can only be detected as a pattern of agent interactions, and can only be resolved by changing that pattern (e.g. by replacing one agent with another that has different input requirements). Agent developers must thus anticipate all the contexts in which the agent may be used, but this is extremely difficult. No systematic methodology is available, however, to help developers identify all relevant exception types and resolution strategies. Making changes in exception handling behavior is difficult because it potentially requires coordinated changes in multiple agents. Agents become much harder to maintain, understand and reuse because the relatively simple normative behavior of an agent becomes obscured by a potentially large body of code devoted to handling exceptional conditions. Finally, it is unrealistic to expect that all agents will have sophisticated exception handling capabilities built in. In many cases we will have to be able to operate with agents whose design incorporates only the most basic capabilities.

A few efforts have done some preliminary exploration of the use of distinct exception handling services. This work has occurred predominantly in the context of business process enactment [1-3, 5, 6], manufacturing control [7-9] and planning [10, 11]. The process enactment and manufacturing work, in general, has either not evolved to the point of constituting a computational model, or has been applied to a very limited range of domains (e.g. just software engineering or flexible manufacturing cell control) and

exception types (e.g. just inappropriate task assignments). The planning work, by contrast, has developed a range of computational models but their ability to redesign a multi-agent work process in response to an exception is contingent upon the planning approach having been used to develop the original work process. This requirement is difficult or impossible to satisfy in an environment where the work process emerges dynamically via the interaction of multiple heterogeneous agents.

3. Our Approach: A Shared Exception Handling Service

Our approach transcends the limitations of current approaches by creating a shared exception handling service that can be "plugged", with little or no customization, into existing agent systems to add the ability to function in exception-prone environments. This service can be viewed as a kind of "coordination doctor"; it knows about the different ways multi-agent systems can get "sick", actively looks system-wide for symptoms of such "illnesses", and prescribes specific interventions instantiated for this particular instance from a body of general treatment procedures. Agents need only implement their normative behavior plus a minimal set of interfaces that assume only that each agent can report on its own behavior and modify its own actions to at least some extent. This vision is realized by building on four key innovations:

- We define a clear division of labor. Problem solving agents focus on executing their own "normal" problem solving behavior, while the exception handling agents focus on detecting and resolving exceptions in the agent ensemble as a whole.
- The exception handling service applies a knowledge base of generic exception handling detection, diagnosis and resolution expertise which can be applied to a wide range of domains.
- The "cost of admission" is only that agents understand a standard language providing at least a basic level of self-awareness and self-modifiability, comparable to what is required of agents capable of reasonably sophisticated coordination in exception-free contexts.
- This service can be implemented as a set of standard agents that can be "plugged" in to any agent system whose agents support the language interfaces described above.

We describe our approach in more details in the paragraphs below.

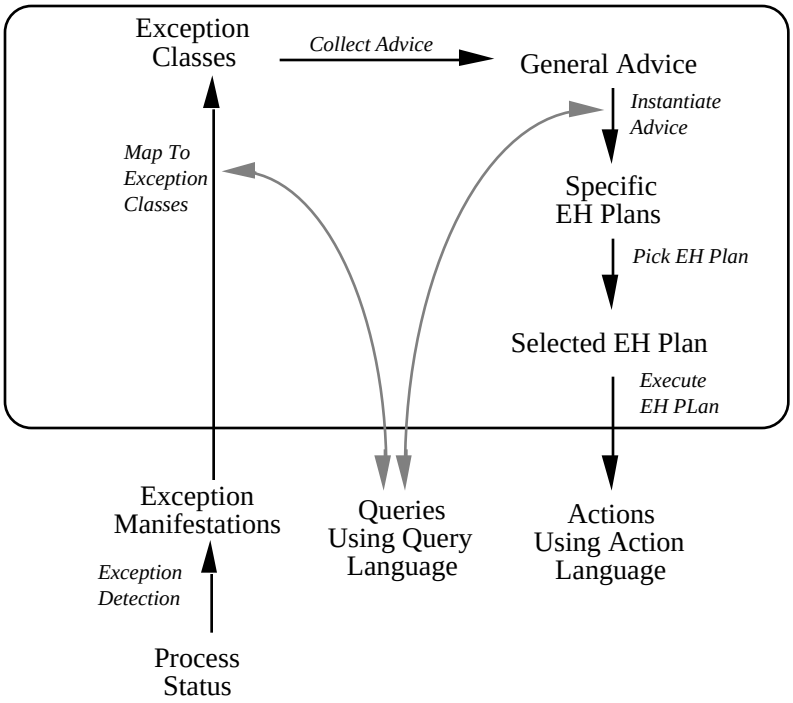
Generic Exception Handling Expertise: The key element underlying our approach is the simple but powerful notion that generic and reusable exception handling expertise can be usefully separated from the knowledge used by agents to do their "normal" work. There is substantial evidence for the validity of this claim. Early work on expert systems development revealed that it is useful to separate domain-specific problem solving and generic control knowledge [12, 13]. Analogous insights were also confirmed in the domains of collaborative design conflict management [14] and in preliminary work on

process exception management [15]. Generic exception management strategies are easy to find [16]. Some examples include:

- if an agent plan has failed, backtrack to a different plan for achieving the same goal
- if a highly serial process is operating too slowly to meet an impending deadline, and the subtasks have only serial dependencies, use pipelining (i.e. releasing results for earlier subtasks before later subtasks are completed) to increase concurrency
- if an agent receives garbled data, trace the problem back to the original source of the faulty data, eliminate all decisions that were corrupted by this error, and start again
- if an agent may be late in producing a time-critical output, see whether the consumer agent will accept a less accurate output in exchange for a quicker response
- if multiple agents are causing wasteful overhead by frequently trading the use of a scarce shared resource, change the resource sharing policy such that each agent gets to use the resource for a longer time
- if a new high-performance resource applicable to a time-critical task becomes available, consider reallocating the task from its current agent to the new agent
- if an agent in a serial production line fails to perform a task, try to re-allocate the task to an appropriately skilled agent further down the line

It is our experience that such strategies are easy to acquire from a wide range of research literature sources, as well as by generalizing from the vast range of exception handling cases we all encounter. We have identified about 300 strategies to date; more details will be given below.

Heuristic Classification: A useful metaphor for organizing such expertise, we have found, is to treat exception handling (EH) as a heuristic classification process [17] analogous to that used in medical diagnosis. In this approach, an exception manifestation (i.e. symptom), once detected, is mapped to candidate diagnoses in a pre-defined taxonomy of generic underlying causes; generic strategies associated with these diagnoses are then instantiated into candidate exception resolution plans, one of which is then selected and executed:



The approach thus instantiates generic exception handling expertise into specific situations. The EH service communicates with agents using pre-defined languages for learning about the exception(s) (the query language) and for describing exception resolution actions (the action language). Agents can take any form as long as they are capable of responding appropriately to at least a minimum subset of these query and action languages.

Exception Detection: The first step in detecting exceptions is, of course, having a model of what the "correct" behavior for the multi-agent system is. When an agent is introduced into a multi-agent system, therefore, it must register at least a rudimentary model of its normative behavior with the exception handling service. This model is mapped to a list of the failure modes that are known to occur for that kind of normative behavior, and sentinels are generated to detect those modes.

Failure mode identification is done making use of a taxonomy of generic problem solving processes wherein each process is annotated with the different ways it can fail. When a new agent is registered, we merely identify the generic processes corresponding to that agent's behavior, and derive the applicable failure modes from that. For example, it is typical for agents to require the output of another agent. The processes for managing such "flow" dependencies need to make sure that the right thing gets to the right place at the right time [18]. This immediately implies a set of possible failure modes including an input being late ("wrong time"), of the wrong type ("wrong thing") and so on. Similar analyses can be done for other generic processes, e.g. resource sharing, diagnosis, synthesis, market-based coordination and so on. We are building for this purpose upon

the process taxonomy being developed in the context of the MIT Process Handbook. The Handbook is a substantive (3700+ entity) and growing repository of coordination mechanisms and other problem solving processes [18-20] which has been under development for roughly the past five years in the MIT Center for Coordination Science.

Our work to date in performing failure mode analysis has revealed a wide range of exception types [15]. Exceptions in general involve violations of some either implicit or explicit assumption underlying a collaborative work process (e.g. stability of resources, correctness of output etc.) and can include change in resources, organizational structure, agent system policies, task requirements or task priority. They can also include incorrectly performed tasks, missed due dates, resource contentions between two or more distinct processes, unexpected opportunities to merge or eliminate tasks, conflicts between actions taken in different process steps and so on.

Every failure mode can have associated with it a script that searches for the pattern of agent behavior corresponding to that failure. These scripts, once instantiated, play the role of "sentinels" that alert the exception handling service when the condition they were created to detect has occurred. A typical sentinel, for example, may check for a task becoming late, violation of resource limits, circular wait patterns, and so on. To define these scripts, we build upon pattern matching tools developed in previous work [21].

Exception Diagnosis: The diagnosis mechanisms works by traversing a taxonomy of possible exception diagnoses based on the presenting symptoms as well as information about the process model being enacted. This is a "shallow model" approach [22] because it is based on compiled empirical and heuristic expertise rather than first principles. This approach is appropriate for domains, such as medical diagnosis, where complete and consistent first-principle-based behavioral models do not exist. An important characteristic of heuristic classification is that the diagnoses represent *hypotheses* rather than guaranteed *deductions:* multiple diagnoses may be suggested by the same symptoms, and often the only way to verify a diagnosis is to see if the associated prescriptions are effective.

The diagnosis hierarchy, in our current model, is structured as a decision tree wherein the system starts at the top most abstract diagnosis and attempts to refine it to more specific diagnoses by traversing down the tree and selecting the appropriate decision branches by asking questions, expressed as query language statements, of the relevant problem solving agents. For example, if the system is assessing whether the diagnosis of "circular wait deadlock" applies, it may ask agents for which other agents they are waiting for inputs from. This traversal can result in more than one candidate diagnosis, since multiple causes may be suggested by the same symptoms.

Exception Resolution: Once one or more candidate diagnoses for an exception have been identified, the next step is to generate, using a knowledge base of generic exception resolution strategies, specific plans for resolving the problem. A diagnosis class will

often have several potential resolution strategies. Since they may not all be applicable for a particular exception, a decision tree procedure identical to that used to select diagnoses is used to find the generic strategies for a given diagnosis. Strategies are represented as executable script templates whose actions are described using the action language. Every template has "slots" which are filled with context-specific values, found using query language queries, to produce specific exception resolution plans. The resolution strategy "backtrack to untried plan for goal", for example, would include slots for the goal and plan that are filled in by asking the affected agent what goal was trying to achieve, and what other plans are available for achieving that same goal. Typically, many possible candidate plans can be generated for a given exception. We can backtrack, for example, in as many ways as there are alternative plans. In our previous work we have found that a relatively small collection of domain-independent heuristics (e.g. "pick the plan that makes the smallest change") has been effective in producing a useful ranking of candidate exception resolution plans.

The Query and Action Languages: As we have seen, the query and action languages represent the medium by which the exception handling service interacts with the problem solving agents to detect, diagnose and resolve exceptions. The query language is used to get agent state information, and the action language is used to modify it.

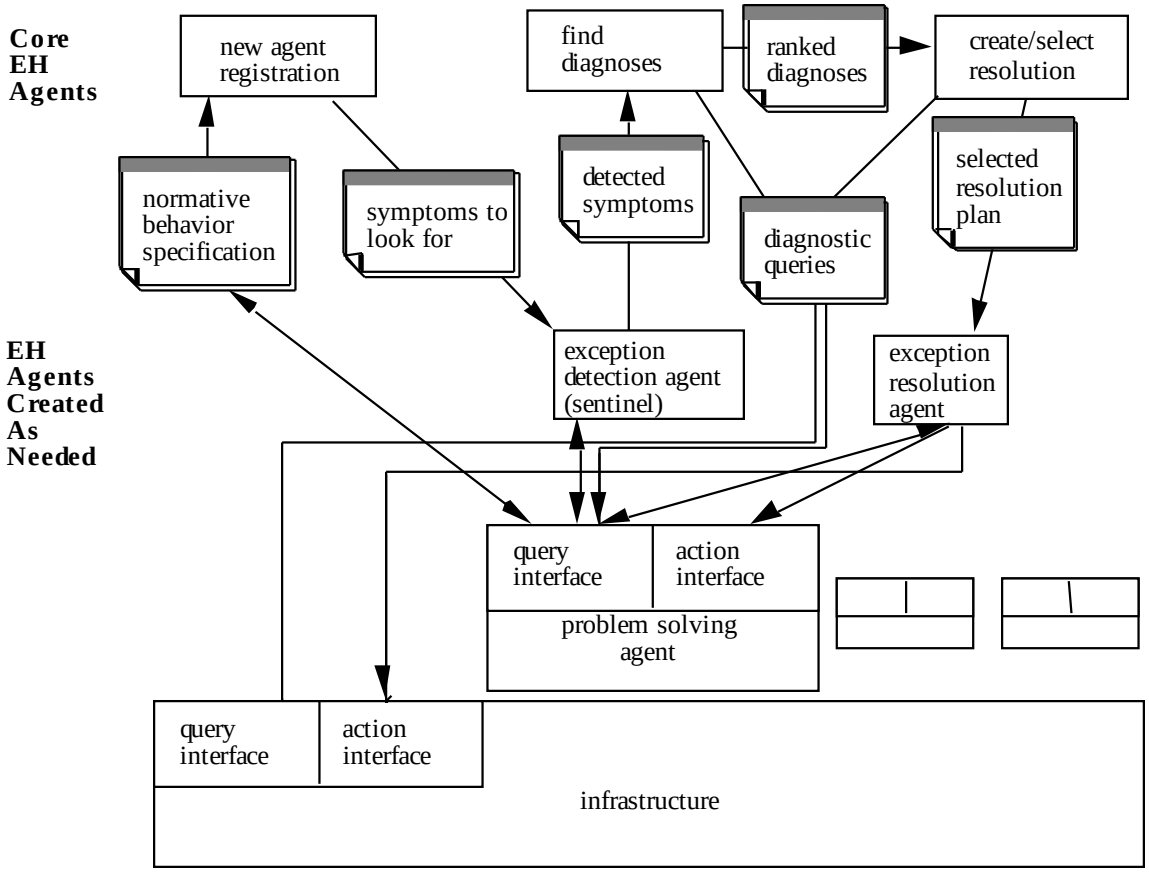
The query language we use builds upon that developed in earlier systems [16, 23] extending it to include queries concerning normative agent behavior models. The query language is relatively large, and we will make the effort to consolidate it into a smaller set of critical query types. The action language, in contrast, consists of a relatively small set of operators [16]. These include changing the process model (reordering, deleting or adding new tasks; changing the resources allocated to a task; canceling tasks) and changing the work package contents.

Our experience to date has shown that agents do not have to understand all of the query or action language primitives in order to benefit from the exception handling service, but the more primitives they can understand, the more effective the exception handling service is likely to be. This is because the more generic exception diagnoses and resolution strategies tend to require only the simplest and easiest to implement queries and actions, but the more sophisticated (and presumably more effective) diagnoses and resolutions use the more "advanced" primitives.

The query and action languages can be viewed as representing a "price of admission" to our approach. These languages only require, however, that the individual problem solving agents be able to describe their own behavior as well as a modify their own actions; the exception handling service is responsible for understanding how local knowledge and actions can be coordinated to produce a globally effective exception response. Previous DAI research suggests, moreover, that for many cases we want our agents to have

roughly that level of self-awareness and self-modifiability in order to support effective coordination even in the absence of exceptions [24].

System Architecture: The capabilities described above can be implemented straightforwardly as agents that can simply be plugged-in to an existing agent system with suitable interfaces:



This architecture consists of exception handling agents, problem solving agents as well as the agent systems infrastructure, all of which must support at least the base level query and action languages. When a new agent is created, the “new agent registration” agent takes a description of its normative behavior and creates sentinels (exception detection agents) as necessary to look for evidence of dysfunctional behavior. Should a sentinel detect such symptoms, this information is sent to a “diagnosis” agent which produces a ranked set of candidate diagnoses. These in turn are sent to the resolution agent which defines a resolution plan instantiated in the form of a “resolution” (exception resolution) agent. We can have redundant copies of these agents, thereby increasing performance and addressing potential failures in the exception handling agents themselves.

Human in the Loop: While the architecture above has been presented as a fully automated one, in at least some cases it will make sense to include a human “executive manager” in

the loop. Our previous work in this area, for example, used human input to modify the ranking of diagnoses and resolution plans proposed by the exception handling service, and thereby direct the system in the direction the human users considered more appropriate [25]. We have found that the exception handling service can help human users better understand and more creatively resolve exceptions, even if they did not use the particular resolutions proposed by the system.

4. Evaluation: Contribution to Improving Agent-Based Systems

The ideas described in this paper have already been substantially validated through nearly a decade of development and evaluation of successful systems for resolving multi-agent exceptions in the collaborative design [14, 16, 26] and collaborative requirements capture [21] domains. This led to the development of the basic heuristic classification approach, software tools for exception diagnosis and resolution, a substantive standardized language for communication between agents and the exception handling service, a highly expressive rationale capture language [23], as well as a substantive and growing knowledge base of exception resolution heuristics. More recent work has begun applying these ideas to a broader range of exception types [15, 27, 28]. The current contents of the exception handling knowledge base can be characterized as follows:

Aspect	*Number*	*Examples*
conflict detection strategies	~10	· check for violated resource budget · check for inconsistent parameter constraints
query operators in standardized agent communication language	~100	· what is the rationale for the decision? · is the parameter constraint relaxable?
action operators in standardized agent communication language	~10	· relax constraint · try different plan for goal
exception diagnoses	~100	· agent constraints too ambitious · excessive serialization in work process
exception resolution strategies	~300	· relax constraints, maximizing summed utilities · pipeline tasks with serial dependencies
exception plan selection heuristics	~10	· pick most specific resolution plan · abandon low level goals before high level goals

Our results to date suggest that the exception handling service approach enables two classes of important benefits:

- easier agent development: This approach makes it much easier to develop, understand, maintain and reuse problem solving agents, since developers can focus on their normative behavior without having to build in responses to all possible exceptions. This greatly reduces the cost of achieving the transparent agent interoperability that underlies the appeal of agent systems. Another advantage is that this approach does not rely on the existence of powerful exception-handling support facilities in every agent's implementation language.
- easier to specify effective exception handling behavior: We are less likely to miss important failure modes, and will probably use better exception resolution practices, by taking advantage of a systematically accumulated knowledge base of exception handling "best practices". It will also be much easier, we believe, to specify and modify systemic exception-handling expertise if it is treated as a functional unit rather than captured as a series of carefully designed interlocking behaviors spread over myriads of diverse agents.

These benefits translate into more reliable, predictable and efficient agent-based system operation.

5. Future Work

We plan to follow two inter-related tracks in our future work: (1) further development of the exception handling knowledge base and underlying diagnostic technology, and (2) further evaluation of this technology in both simulated and "real-world" testbed contexts.

Technical issues we currently consider important include extending the diagnostic approach to be able to handle multiple simultaneous exceptions in a coordinated way [29], as well as reducing as much as possible the size of the query/action languages that agents need to understand in order to interact effectively with the exception handling service. We also plan to explore "model-based" diagnostic approaches [30, 31] which have been applied with good results to explaining faults in that subclass of systems where complete behavioral models exist

Our evaluation plan consists of a graded series of experiments, occurring first in a simulated agents testbed (where we have the maximum flexibility in designing the test scenarios), and then transitioning to an externally developed testbed (to assess and demonstrate the ability to extend a pre-extending agent system with our exception handling technology). The simulated testbed will evaluate agent system behavior using such heuristics as problem solving time, effectiveness of resource utilization, the understandability of the agent ensemble behavior to human observers, ability of problem solving agents to work in multiple ensemble contexts w/o modification, and the ability to control the tradeoff between exception handling and problem solving effort. We currently

plan to do our first tests in the manufacturing logistics domain. The second testbed will enable a "technology integration experiment" wherein we explore integration of our exception handling technology into a agent system not originally designed for that purpose. This will allow us to assess and demonstrate the ability of our technology to be "plugged in" to other testbeds, help us identify the knowledge base and query/action language enhancements needed, if any, and provide insights into how integration can best be done. We can therefore view this as a "final rehearsal" for adoption of our technology by agent system developers outside of our project team. We are currently considering, for this purpose, the MIT AI Lab's "Intelligent Room", a large real-time agent-based information gathering/presentation system [32-34].

6. Acknowledgements

The authors gratefully acknowledge the support of the DARPA CoABS Program (contract F30602-98-2-0099) while preparing this paper.

7. References

[1] Auramaki, E. and M. Leppanen. Exceptions and office information systems. in Proceedings of the IFIP WG 8.4 Working Conference on Office Information Systems: The Design Process. 1989. Linz, Austria.

[2] Karbe, B.H. and N.G. Ramsberger, Influence of Exception Handling on the Support of Cooperative Office Work, in Multi-User Interfaces and Applications, S. Gibbs and A.A. Verrijin-Stuart, Editors. 1990, Elsevier Science Publishers. p. 355-370.

[3] Strong, D.M., Decision support for exception handling and quality control in office operations. Decision Support Systems, 1992. 8(3).

[4] Suchman, L.A., Office Procedures as Practical Action: Models of Work and System Design. ACM Transactions on Office Information Systems, 1983. 1(4): p. 320-328.

[5] Mi, P. and W. Scacchi. Articulation: An Integrated Approach to the Diagnosis, Replanning and Rescheduling of Software Process Failures. in Proceedings of 8th Knowledge-Based Software Engineering Conference. 1993. Chicago, IL, USA: IEEE Comput. Soc. Press; Los Alamitos, CA, USA.

[6] Kreifelts, T. and G. Woetzel. Distribution and Error Handling in an Office Procedure System. in IFIP WF 8.4 Working Conference on Methods and Tools for Office Systems. 1987. Pisa Italy.

[7] Visser, A., An exception-handling framework. International Journal of Computer Integrated Manufacturing, 1995. 8(3): p. 197-203.

[8] Parthasarathy, S. Generalised process exceptions-a knowledge representation paradigm for expert control. in Proceedings of the Fourth International Conference on

the Applications of Artificial Intelligence in Engineering. 1989. Cambridge, UK: Comput. Mech. Publications; Southampton, UK.

[9] Katz, D.M., S. Exception management on a shop floor using online simulation. in Proceedings of 1993 Winter Simulation Conference - (WSC '93). 1993. Los Angeles, CA, USA: IEEE; New York, NY, USA.

[10] Birnbaum, L., et al. Model-Based Diagnosis of Planning Failures. in AAAI-90. 1990.

[11] Broverman, C.A. and W.B. Croft. Reasoning About Exceptions During Plan Execution Monitoring. in AAAI-87. 1987.

[12] Gruber, T.R., A Method For Acquiring Strategic Knowledge. Knowledge Acquisition, 1989. 1(3): p. 255-277.

[13] Barnett, J.A., How Much Is Control Knowledge Worth? A Primitive Example. Artificial Intelligence, 1984. 22(1): p. 77-89.

[14] Klein, M., Supporting Conflict Resolution in Cooperative Design Systems. IEEE Systems Man and Cybernetics, 1991. 21(6): p. 1379-1390.

[15] Klein, M., Exception Handling in Process Enactment Systems, . 1997, MIT Center for Coordination Science: Cambridge MA.

[16] Klein, M., Conflict Resolution in Cooperative Design, in Computer Science. 1989, University of Illinois: Urbana-Champaign, IL.

[17] Clancey, W.J., Classification Problem Solving. Aaai, 1984: p. 49-55.

[18] Malone, T.W. and K.G. Crowston, The interdisciplinary study of Coordination. ACM Computing Surveys, 1994. 26(1): p. 87-119.

[19] Dellarocas, C., et al. Using a Process Handbook to Design Organizational Processes. in Proceedings of the AAAI 1994 Spring Symposium on Computational Organization Design. 1994. Stanford, California.

[20] Malone, T.W., et al. Tools for inventing organizations: Toward a handbook of organizational processes. in Proceedings of the 2nd IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises (WET ICE). 1993. Morgantown, WV, USA.

[21] Klein, M., An Exception Handling Approach to Enhancing Consistency, Completeness and Correctness in Collaborative Requirements Capture. Concurrent Engineering Research and Applications, 1997(March).

[22] Chandrasekaran, B. and S. Mittal, Deep Versus Compiled Knowledge Approaches To Diagnostic Problem Solving. Int. J. Man-Machine Studies, 1983: p. 425-436.

[23] Klein, M., Capturing Design Rationale in Concurrent Engineering Teams. IEEE Computer, 1993. 26(1): p. 39-47.

[24] Findler, N.V. and R. Lo, An Examination of Distributed Planning in the World of Air Traffic Control, in Readings in Distributed Artificial Intelligence, A.H. Bond and L. Gasser, Editors. 1988, Morgan Kaufmann: California. p. 617--627.

[25] Klein, M. and S.C.-Y. Lu. Insights Into Cooperative Group Design: Experience With the LAN Designer System. in Sixth International Conference on Applications of Artificial Intelligence in Engineering (AIENG '91). 1991. Uk.

[26] Klein, M., Supporting Conflict Management in Cooperative Design Teams. Journal on Group Decision and Negotiation, 1993. 2: p. 259-278.

[27] Klein, M., Conflict Management as Part of an Integrated Exception Handling Approach. AI in Engineering Design Analysis and Manufacturing (AI EDAM), 1995. 9: p. 259-267.

[28] Klein, M., Core Services for Coordination in Concurrent Engineering. Computers in Industry, 1996.

[29] Wu, T.D. Efficient Diagnosis of Multiple Disorders Based on a Symptom Clustering Approach. in Proceedings of AAAI-90. 1990.

[30] Genesereth, M.R. Diagnosis Using Hierarchical Design Models. in Proceedings of AAAI-82. 1982.

[31] Kleer, J.d., A.K. Macworth, and R. Reiter. Characterizing Diagnoses. in Proceedings of AAAI-90. 1990.

[32] Kautz, H., et al. An Experiment in the Design of Software Agents. in Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94). 1994. Seattle, WA.

[33] Coen, M. Building Brains for Rooms: Designing Distributed Software Agents. in Proceedings of IAAI-97. 1997.

[34] Coen, M. Towards Interactive Environments: The Intelligent Room. in Proceedings of HCI-97. 1997.